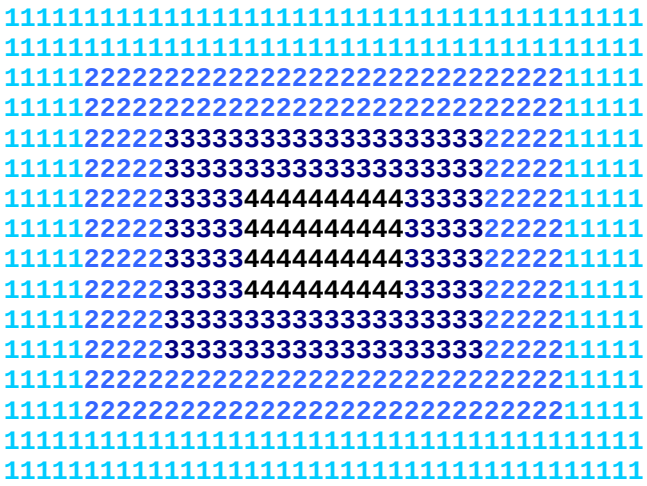
CHARLES ASHBACHER

Smarandache Sequences, Stereograms and Series



Smarandache Stereogram

Hexis
Phoenix
2005

Charles Ashbacher
Mount Mercy College

Smarandache Sequences, Stereograms and Series

Hexis
Phoenix
2005

This book can be ordered in a paper bound reprint from:

Books on Demand
ProQuest Information & Learning
(University of Microfilm International)
300 N. Zeeb Road
P.O. Box 1346, Ann Arbor
MI 48106-1346, USA
Tel.: 1-800-521-0600 (Customer Service)
http://wwwlib.umi.com/bod/search/basic

Peer Reviewers:
Henry Ibstedt, Issy les Moulineaux, France
Amarnath Murthy, Gujarat, India
Lamarr Widmer, Messiah College, Grantham PA USA

Many books can be downloaded from the following
E-Library of Science:
http://www.gallup.unm.edu/~smarandache/eBooks-otherformats.htm

ISBN: 1-931233-23-3

Standard Address Number: 297-5092
Printed in the United States of America

Preface

This is the fifth book that I have written that expands on the ideas of Florentin Smarandache. In addition, I have edited two others that also deal with the areas of mathematics under the Smarandache Notions umbrella. All of this is a credit to the breadth and depth of his mathematical achievement. Therefore, I once again must commend and thank him for providing so much material to work with. I also would like to thank J. McGray for her encouragement and patience as I struggled to make this book a reality. The material cited in this book can be found at the website http://www.gallup.unm.edu/~smarandache/.

The deepest thanks go to my mother Paula Ashbacher, who encouraged me to play sports, but in the off chance that I would never learn to hit the curve ball, also insisted that I read books. This proved to be a wise career strategy.

Finally, I would like to express my deep love for Kathy Brogla, my partner/soul mate/best friend. So pretty and vivacious, she makes life fun, exciting and a joy to experience every single day. She is a remarkable woman and I am so blessed to have her in my life. Kathy is also the creator of the image on the front cover.

Dedication

To my lovely daughter Katrina, she will always be first in my queue.

Table Of Contents

Chapter 1

Sequences Formed By Concatenating All the Positive Integers

F. Smarandache has defined many sequences of integers formed by concatenating the positive integers in specific ways. A question that is often asked about most of these sequences concerns the number of primes in the sequence. As Paul Erdos pointed out, determining how many primes are in sequences of this type is a problem that the current state of mathematical knowledge is unprepared to solve. In general, the sequences grow rather fast, so after a very short time, the terms are too large for modern computers to determine if they are prime. The only hope is to find some property of the elements of the sequence that can be used to very quickly determine if it can be factored.

In this chapter, we will cover the sequences that Smarandache defined where the nth element is constructed by concatenating all of the positive integers up through n. Each of these sequences is defined, and then a Java program that computes the elements of the sequence is given. Where applicable, a small search for primes is performed and some properties of the sequences are proven.

The programs are offered up as freeware, to be used by anyone who wishes to do some exploration on their own. All of the programs rely on the BigInteger class defined in the java.math package of Java. It is an extended precision data type that allows for very large numbers to be represented and manipulated. The only limitations to using them are system memory and personal patience.

1) Consecutive Sequence:

1,12,123,1234,12345,123456,1234567,12345678,123456789,12345678910,
1234567891011,123456789101112,12345678910111213,...

In this case, the elements of the sequence are formed by concatenating the next integer to the right of the previous element of the sequence. Smarandache asked the question:

How many primes are there among these numbers?

This is an unsolved problem, although clearly they will be very rare.

Smarandache continues the problem by making the statement, "In a general form, the Consecutive Sequence is considered in an arbitrary numeration base B."

For example if we use base 3, the elements of the sequence are

1, 12, 1210, 121011, 12101112, 1210111220, . . .

which are equivalent to the following decimal numbers

1, 5, 48, 436, 3929, 35367, 318310, 2864798, . . .

The first elements of the sequence for base 4 are

1, 12, 123, 12310, 1231011, 123101112, 12310111213, 1231011121320, . . .

which are equivalent to the following decimal numbers.

1, 6, 27, 436, 6981, 111702, 1787239, 28595832, 457533321, . . .

The first few elements of the sequence for base 5 are

1, 12, 123, 1234, 123410, 12341011, 1234101112, 123410111213, . . .

which are equivalent to the following decimal numbers

1, 7, 38, 194, 4855, 121381, 3034532, 75863308, 1896582709,. . .

The first few elements of the sequence for base 6 are

1, 12, 123, 1234, 12345, 1234510, 123451011, 12345101112, 1234510111213, . . .

which are equivalent to the following decimal numbers

1, 8, 51, 310, 1865, 67146, 2417263, 87021476, . . .

The first few elements of the sequence for base 7 are

1, 12, 123, 1234, 12345, 123456, 12345610, 1234561011, 123456101112, . . .

which are equivalent to the following decimal numbers

1, 9, 66, 466, 3267, 22875, 1120882, 54923226, . . .

The first few elements of the sequence for base 8 are

1, 12, 123, 1234, 12345, 123456, 1234567, 123456710, 12345671011, . . .

which are equivalent to the following decimal numbers

1, 10, 83, 668, 5349, 42798, 342391, 21913032, 1402434057, . . .

The first few elements of the sequence for base 9 are

1, 12, 123, 1234, 12345, 123456, 1234567, 12345678, 1234567810, 123456781011, . . .

which are equivalent to the following decimal numbers

1, 11, 102, 922, 8303, 74733, 672604, 6053444, 490328973, 1061941159, . . .

Using the bases from three through nine to form new sequences, we can then ask the same question regarding the number of primes in the sequence. That appears to be a question just as difficult as the one for base ten numbers, and is summarized in the following statement.

Problem: Which of the sequences formed by right concatenating the positive integers in the bases 3 through 10 contains the largest number of primes?

A computer program was written that searched through the first 500 elements of each of the consecutive sequences in the bases three through ten looking for primes, and that program appears in listing 1.

Listing 1

```
/* Program to construct the terms of the Consecutive Sequence for an arbitrary base and
   determine which terms are prime. Written by Charles Ashbacher. */

import java.math.*;
import java.io.*;

public class BaseNPrime
{
// The upper limit to the test for primeness
 static final int UPPERLIMIT=500;
// This is the base used to generate the numbers in the Consecutive Sequence
// Change it for other bases.
 static BigInteger base1=new BigInteger("3");
// File where the data is sent
 static String filename="Basevalues.txt";
 static File output;
 static BufferedWriter out;

/* This function converts the first input BigInteger into the String equivalent as
    expressed in the second BigInteger base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
```

```
 {
  theRemainder=theNumber.mod(theBase);
  theReturn=theRemainder.toString()+theReturn;
  theNumber=theNumber.divide(theBase);
 }
 return theReturn;
}

/* This function converts the input String into the decimal number equivalent using the
    BigInteger input as the base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                          BigInteger  theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }

 public static void main(String[] args)
 {
  String str1;
  BigInteger theCount=new BigInteger("2");
  String base1String=new String("1");
  BigInteger base1Value;
  String theConversion;
  boolean primetest;
  System.out.println("The base is "+base1);
  try
  {
   output=new File(filename);
   output.createNewFile();
   if(!output.isFile())
   {
    System.out.println("Error creating the file");
    IOException ioe=new IOException();
    throw ioe;
   }
```

```
   out=new BufferedWriter(new FileWriter(output));
   out.write("The base is "+base1);
  }
  catch(IOException ioe)
  {
   System.out.println("Cannot open the designated file");
   System.exit(0);
  }
  for(int i=1;i<=UPPERLIMIT;i++)
  {
   theConversion=ConvertToBase(theCount,base1);
   base1String=base1String+theConversion;
   base1Value=ConvertToDecimal(base1String,base1);
   primetest=base1Value.isProbablePrime(100);
   if(primetest)
   {
    str1="Index number "+theCount+" is prime\n";
    str1=str1+base1String+"\n";
    str1=str1+base1Value+"\n";
    System.out.println(str1);

// Write the data to a file.
    try
    {
     out.write(str1);
    }
    catch(IOException ioe)
    {
     System.out.println("Could not write text to the file");
    }
   }
   theCount=theCount.add(BigInteger.ONE);
  }
  System.out.println("The search was through "+UPPERLIMIT);

  try
  {
   out.write("The search was through "+UPPERLIMIT);
   out.close();
  }
  catch(IOException ioe)
  {
   System.out.println("Could not close the file");
  }
 }
}
```

The program uses the BigInteger data type, which can hold numbers of arbitrary size, and the function

base1Value.isProbablePrime(100);

returns a true if the number is prime to a probability less than 1 divided by 2^{100}.

The results of the searches are summarized in table 1.

Table 1

Base	Element numbers that are prime
3	2, 5, 82
4	None
5	2, 113, 162
6	11,173
7	10,37
8	3
9	2,14
10	None

The most interesting thing about this table is that there are no primes for the base 4 and 10 sequences and only one small one for the base 8. To further pursue the matter, the program was rerun for the first 1000 elements of the base 4 sequence and no primes were found.

Problem: Are there any primes in the base four sequence?

Another interesting question arises when we see that the fourth term of the base 3 sequence (121011) is decimal 436, which is the same decimal value as the fourth term of the base 4 sequence (12310).

Problem: With the exception of the obvious value of 1, are there any other cases where the decimal value of the nth term in one base is the same as the decimal value of the nth term in another base?

A computer program was written to test this problem for all bases 3 through 9 and no other solutions were found. The program was modified to check for circumstances where the relative sizes of the terms of the matched elements of the sequences changed and this happens rather frequently. In other words, there are many cases where $s_k < t_j$ and $s_{k+1} > t_{j+1}$ for s_i and t_i the sequences for two different bases. Therefore, it is possible that there are other circumstances where the elements with the same index in two different sequences are the same.

The program was then modified to search for terms where the nth term of the sequence for one base is equal to the kth term of the sequence for another base. It was run for the first three hundred terms for all pairs of bases from three through ten. The only solution found was element number 4 of the base three sequence being equal to element number four of the base four sequence.

Problem: Is there any other case where an element of a base B_1 Consecutive Sequence is equal to an element of another base B_2 sequence?

However, there is one simple pattern of these sequences that is easy to verify.

Theorem: If B > 2 is an arbitrary base and we form the sequence of numbers

1, 12, 123, . . . , 123 . . .(B-1), 123 . . . (B-1)10, . . .

and let m represent the nth element of the sequence, we have the following pattern.

If B is odd, then

m is odd if n is congruent to 1 or 2 modulo 4
m is even if n is congruent to 0 or 3 modulo 4.

If B is even, then

m is odd if n is congruent to 1 or 3 modulo 4
m is even if n is congruent to 0 or 2 modulo 4.

Proof:
In both cases, the proof is by induction

Case 1: The base B is odd.

Basis step

For B = 3, the first four terms are

1, 12, 1210, 121011

which are decimal

1, 5, 48, 436 respectively.

For B > 3, the first four terms are

1, 12, 123, 1234,

which, when expressed as an algebraic formula , are

$1, B + 2, B^2 + 2*B + 3, B^3 + 2*B^2 + 3*B+ 4.$

The first term is odd, and since B is odd, the odd B plus the even 2 is odd. With B odd, all powers of B are also odd, so the third term is the sum of two odds and an even, which is even. The fourth term is the sum of two odds and two evens, which is even.

Inductive step

Take the nth element where n is congruent to 1 modulo 4, which means that n is also odd and assume that this element is odd.

To form the next element, multiply the nth element by a power of B, corresponding to how many digits there are in the next number to be concatenated on the right. Since this power of B is odd and the original number is odd, the product will be odd. Since n was odd, n+1 is even, and we will add the base B equivalent of n+1 to form the next element. With the product odd we have added an even to an odd, which is odd.

To form the next element, we multiply the odd n+1 element by a power of B, yielding an odd product. Since n was odd, n+2 is also odd, so we are adding an odd to an odd, making an even.

To form the next element, we multiply the even n+2 element by a power of B, which would be even. Since n was odd, n+3 is even, and we are adding an even number to an even number, which is an even number.

To form the next element, we multiply the even n+3 element by a power of B, yielding an even product. Since n was odd, n + 4 is also odd, so we are adding an odd to an even, which is odd.

We started with n congruent to 1 modulo 4, so n+1 would be congruent to 2 modulo 4, n+2 would be congruent to 3 modulo 4, n+3 would be congruent to 0 modulo 4 and n+4 would be congruent to 1 modulo 4. This completes the cycle.

Case 2: The base B is even.

Basis step

For B = 4, the first four terms are

1, 12, 123, 12310

which are decimal

1, 6, 27, 436.

For B > 4, the first four terms are

1, 12, 123, 1234

which, expressed as algebraic formulas , are

1, B + 2, B^2 + 2*B + 3, B^3 + 2*B^2 + 3*B+ 4.

The first is clearly odd and the second is an even plus an even. Since all powers of B are even, the parity of the numbers will be based on the parity of the last term, which is odd for 3 and even for 4.

Inductive step

Take the nth element where n is congruent to 1 modulo 4, which means that n is also odd and assume that this element is odd.

To form elements n+1, n+2, n+3 and n+4, we will be multiplying the previous term by a power of B, which is always even. Therefore, the parity of the next term will be based on the parity of the term number, which is even for n+1, odd for n+2, even for n+3 and odd for n+4. This completes the cycle and the inductive condition is verified.

2) Symmetric Sequence:

1,11,121,1221,12321,123321,1234321,12344321,123454321,1234554321,
12345654321,123456654321,1234567654321,12345677654321,123456787654321,
1234567887654321,12345678987654321,123456789987654321,
1234567891098765432 1,12345678910987654321,1234567891011098765432 1,
12345678910111110987654321,...

For this sequence, the next term is constructed by concatenating the positive integers so that they satisfy the following pattern:

If n is odd, form the even number n+1, divide it by two [(n+1)/2 = m] and perform the concatenation

123 . . . (m-1)m(m-1) . . . 321.

If n is even, divide it by 2 [n/2 = m] and perform the concatenation

123 . . . (m-1)mm(m-1). . . 321.

Smarandache then asked the question :

How many primes are there among these numbers?

This is an unsolved problem, although clearly they will also be rare. However, they may not be as rare as the primes in the consecutive sequence, for all the numbers in this sequence are odd.

Smarandache continues the problem by making the statement, "In a general form, the Symmetric Sequence is considered in an arbitrary numeration base B."

For example, if the base is 3, the first few elements of the sequence are:

1, 11, 121, 1221, 121021, 12101021, 1210111021, 1210111021, 12101112111021

which are equivalent to the following decimal numbers

1, 4, 16, 52, 439, 3922, 35350, 318202, 2864599

the last of which is prime.

The elements of the sequences for bases 4 through 9 are computed in a similar way.

A computer program was written that searched through the first 500 elements of each of the symmetric sequences in the different bases looking for primes. This program appears in listing 2.

Listing 2

```
/* This program is used to construct the elements of the symmetric sequences for an
   arbitrary  base, which is stored in the base1 variable. All elements will be sequentially
   constructed up to the element number stored in UPPERLIMIT. Written by Charles
   Ashbacher .*/

import java.math.*;
import java.io.*;

public class SymmetricSequencePrime
{
// The upper limit on the number of elements.
 static final int UPPERLIMIT=500;
// The base used in the construction of the elements.
 static BigInteger base1=new BigInteger("3");
 static String filename="Symvalues.txt";
 static File output;
 static BufferedWriter out;

/* Converts the first input BigInteger into the String representation using the
  second input BigInteger as the base. */
```

```
 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }

/* Converts the String input into the decimal number using the BigInteger input as the
   base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }

 public static void main(String[] args)
 {
  String str1;
  BigInteger theCount=new BigInteger("2");
  BigInteger elementCount=new BigInteger("3");

/* The elements will first be created as strings. The base1String variable will store the
   left side of the element and the base2String variable will store the right side. They will
   be concatenated into sysmString, which will then be converted into the base 10 decimal
   equivalent. */

  String base1String=new String("1");
  String base2String=new String("1");
```

```
BigInteger base1Value;
String theConversion1;
boolean primetest;
int theSwitch=0;
String symString1;
System.out.println("The base is "+base1);
try
{
 output=new File(filename);
 output.createNewFile();
 if(!output.isFile())
 {
  System.out.println("Error creating the file");
  IOException ioe=new IOException();
  throw ioe;
 }
 out=new BufferedWriter(new FileWriter(output));
}
catch(IOException ioe)
{
 System.out.println("Cannot open the designated file");
 System.exit(0);
}

for(int i=1;i<=UPPERLIMIT;i++)
{
 theConversion1=ConvertToBase(theCount,base1);
 if(theSwitch==1)
 {
  base2String=theConversion1+base2String;
  theSwitch=0;
  theCount=theCount.add(BigInteger.ONE);
 }
 else
 {
  base1String=base1String+theConversion1;
  theSwitch++;
 }
 symString1=base1String+base2String;
 base1Value=ConvertToDecimal(symString1,base1);

 primetest=base1Value.isProbablePrime(100);
 if(primetest)
 {
  str1="Index number "+elementCount+" is prime\n";
  str1=str1+symString1+"\n";
```

```
   str1=str1+base1Value+"\n";
   System.out.println(str1);
   try
   {
    out.write(str1);
   }
   catch(IOException ioe)
   {
    System.out.println("Could not write text to the file");
   }
  }
   elementCount=elementCount.add(BigInteger.ONE);
 }
 System.out.println("The search was through "+UPPERLIMIT);

 try
 {
  out.write("The search was through "+UPPERLIMIT);
  out.close();
 }
 catch(IOException ioe)
 {
  System.out.println("Could not close the file");
 }
}
}
```

The results are summarized in table 2.

Table 2

Base	Element numbers that are primes
3	5, 9
4	2, 7
5	None
6	2, 11, 12, 77, 207
7	493
8	144, 226
9	17
10	2, 19, 20

Once again, the testing yields few primes, one base where none were found and there is no discernable pattern to the primes that are found.

However, one fact regarding the presence of primes in these sequences is easy to prove.

Theorem: If p is an odd prime in the base 3 symmetric sequence, then the index must be of the form 4k+1.

Proof: We will actually prove that the elements in the symmetric sequence follow the following pattern.

The element is odd if the index is congruent to 1 modulo 4.
The element is even if the index is congruent to 2, 3 or 0 modulo 4.

Also note that the odd indices are when a new number is included in the middle and the even indices are when the second instance of the number is added.

Basis step

Satisfied since the first four elements (in decimal) are 1, 4, 16 and 52.

Inductive step

Assume that element j in the symmetric list with base 3, where j is congruent to 1 modulo 4, is odd. For reference purposes, we will call it m.

Let r be the largest integer in the list of integers concatenated to form m. Therefore, m will have the form

12 . . .(r-1)r(r-1) . . . 21.

Applying the definition of the sequence, r has the value, (4j+1+1)/2 or 2j+1. Therefore, r is odd. Splitting the number into segments, we have

$(12 \ldots (r-1)) * 3^s + r*3^t + (r-1) \ldots 21$

which has odd parity.

The next element of the sequence is formed by including an additional element into the sum

$(12 \ldots (r-1)) * 3^s + r*3^t + r*3^w + (r-1) \ldots 21$

where the s and t exponents on the threes in the first two segments are now larger.

However, since 3 is odd, the parities of both segments will not change. Since r is odd, the additional term $r*3^w$ is also odd. Therefore, we have three segments whose sum is of odd parity and we added an additional odd term. This changes the parity to even.

The next element is formed by including an additional element (r+1), which is known to be even.

12 . . . r(r+1)r . . . 21.

Written as a sum of products, it is

$(12 \ldots r)*3^s + (r+1)*3^t + (r \ldots 21)$

and since r+1 is even, $(r+1)*3^t$ is also even. Multiplying the element $(12 \ldots r)*3^s$ by additional powers of three will not change the parity of the term. Therefore, we have two terms whose combined parity is even and we are adding an even to them, so the sum is even.

The next term in the sequence has the form

$(12 \ldots r)*3^s + (r+1)*3^t + (r+1)*3^w + (r \ldots 21)$

where the s and t exponents are larger than they were for the previous term. However, multiplying by powers of three does not change the parity of the terms, so the end result is we are adding an even to an even, which is even.

The next term in the sequence would be formed by inserting r+2 into the middle of the previous term to form

12 . . . r(r+1)(r+2)(r+1)r . . . 21

Split this into the three components,

$12 \ldots r(r+1)*3^s + (r+2)*3^t + (r+1)r \ldots 21$

where s and t again have different values than they did in the previous term. Once again, multiplying by powers of three does not change the parity of the number, so the parities of the first and last segments are unaltered. Since we know that r+2 is odd, $(r+2)*3^t$ is also odd and in terms of parity, we are adding an odd to an even, which is odd.

Since this term is now back to the form 4k+1, we have satisfied the inductive step and the theorem is proven.

Corollary : If p is an odd prime in the base B (B odd) symmetric sequence, then it must be of the form 4k+1.

Proof: The proof of the previous theorem relied on the fact that all powers of the base 3 are odd. Since all powers of odd numbers are odd, a similar proof can be constructed for all odd bases.

The next point of interest is to ask if there are elements with the same index in the sequences for two different bases where the values are equal.

Problem: With the exception of the obvious value of 1, are there any other cases where the decimal value of the nth term in the symmetric sequence for one base is the same as the decimal value of the nth term in another base?

A computer program was written to examine this question for all elements of the sequences up through index 1000. No instances of equality were found.

We can also ask if two sequences can have equal elements when the indices are different.

Problem: Is it possible for sequence elements in different bases to have the same value when the indices are different?

This problem was not investigated.

3) Mirror Sequence:

1, 212, 32123, 4321234, . . .

In this case, the nth element of the sequence is formed by concatenating the integers

n(n-1) . . .32123. . . (n-1)n

Smarandache then asked the question:

How many elements of this sequence are primes?

As was the case with the Consecutive and Symmetric sequences, we would expect primes to be rare. Clearly, if the concatenated number is even, then the number cannot be prime.

Smarandache continues the problem by making the statement, "In a general form, the Mirror Sequence is considered in an arbitrary numeration base B."

A computer program was written that searched through the first 500 elements of each of the mirror sequences in bases three through ten looking for primes. This program appears in listing 3.

Listing 3

```
/* This program will create the elements of the Mirror Sequence for the base stored in the
   variable base1. It will then determine if the number is prime. Written by Charles
   Ashbacher. */

import java.math.*;
import java.io.*;
```

```
public class MirrorSequencePrime
{
// Upper limit on the element number to compute.
 static final int UPPERLIMIT=100;
// The base used to construct the elements of the sequence.
 static BigInteger base1=new BigInteger("3");
 static String filename="Mirrorvalues.txt";
 static File output;
 static BufferedWriter out;

/* Function to convert the BigInteger counter to the equivalent String form in the input
base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }

/* Function to convert the input String into the equivalent BigInteger in the input
    BigInteger base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }
```

```
public static void main(String[] args)
{
 String str1;
 BigInteger theCount=new BigInteger("2");
 final BigInteger theTWO=new BigInteger("2");
 String base1String=new String("1");
 BigInteger base1Value;
 String theConversion1;
 boolean primetest;
 System.out.println("The base is "+base1);
 try
 {
  output=new File(filename);
  output.createNewFile();
  if(!output.isFile())
  {
   System.out.println("Error creating the file");
   IOException ioe=new IOException();
   throw ioe;
  }
  out=new BufferedWriter(new FileWriter(output));
 }
 catch(IOException ioe)
 {
  System.out.println("Cannot open the designated file");
  System.exit(0);
 }
 parity=0;
 for(int i=1;i<=UPPERLIMIT;i++)
 {
  theConversion1=ConvertToBase(theCount,base1);

  base1String=theConversion1+base1String+theConversion1;
  base1Value=ConvertToDecimal(base1String,base1);
   primetest=base1Value.isProbablePrime(100);
   if(primetest)
   {
    str1="Index number "+theCount+" is prime\n";
    str1=str1+base1String+"\n";
    str1=str1+base1Value;
    System.out.println(str1);
    try
    {
     out.write(str1);
    }
    catch(IOException ioe)
```

```
    {
     System.out.println("Could not write text to the file");
    }
   }
   theCount=theCount.add(BigInteger.ONE);
  }
  System.out.println("The search was through "+UPPERLIMIT);

  try
  {
   out.write("The search was through "+UPPERLIMIT);
   out.close();
  }
  catch(IOException ioe)
  {
   System.out.println("Could not close the file");
  }
 }
}
```

The results of these searches are summarized in table 3.

Table 3

Base	Element numbers that are prime
3	2, 4, 19
4	5
5	16
6	None
7	2, 8, 194, 291
8	9, 175
9	2, 85
10	13

One property of the Mirror Sequence in the odd bases is easy to prove.

Theorem: If B > 2 is odd, then all of the elements in the base B Mirror Sequence are odd.

Proof: By induction on the length of the elements.

Basis step

The first two elements are

1, $2*B^2 + B + 2$

which are all odd.

Inductive step

Assume that element number N > 2 is odd. This element can be expressed in the form

$(\text{digits})*B^{k+1} + B^k + (\text{reversedigits})$.

Since B^k is odd, the other two numbers in the sum must have the same parity, either both even or both odd.

The creation of the next term in the sequence will add additional digits to both ends of the previous term. These digits will be the same, so the next element will be

$(N+1)_B*B^w + (\text{digits})*B^r + B^{r-1} + (\text{reversedigits})*B^s + (N+1)_B$

where r > k+1. Since all powers of B are odd,

$(\text{digits})*B^r = (\text{digits})*B^{k+1} * B^{r-k-1}$

will have the same parity as

$(\text{digits})*B^{k+1}$

and

$(\text{reversedigits})*B^s$

will have the same parity as

(reversedigits).

So, those two elements of the sum will continue to have the same parity.

Finally,

$(N+1)_B*B^w$ and $(N+1)_B$

will also have the same parity, as multiplication by an odd number does not alter the parity. With B^{r-1} always odd, we have a sum of five elements where the possible parity values are

even + even + odd + even + even
even + odd + odd + odd + even
odd + even + odd + even + odd
odd + odd + odd + odd + odd

so the sum is odd in all cases.

Theorem: If the base B > 2 is even, then the parity of the elements in the base B Mirror Sequence alternate, with the elements of even index being even and the elements of odd index odd.

Proof: The first two elements are 1 and $2*B^2 + B + 2$, which are clearly even and odd. Since all but the last digit of any further element of the sequence will be multiplied by a power of the even base, the overall parity of the element will be determined by the parity of the last digit.

4) Concatenated Natural Sequence

1, 22, 333, 4444, 55555, 666666, 7777777, 88888888, 999999999, 10101010101010101010, . . .

For any natural number n, the nth term is formed by concatenating n n times.

Clearly it is not possible for any element of this sequence to be prime, in fact, it is a modification of the set of repunit numbers.

1, 11, 111, 1111, 11111, 111111, 1111111, . . .

A set of additional sequences can be formed by constructing the sequence using another base.

For example, the first few elements of the base 3 concatenated natural sequence are

1, 22, 101010, 11111111, 1212121212, 202020202020, 21212121212121, . . .

The code of listing 4 will generate the elements of the Natural Sequence for the base value assigned to the variable base1.

Listing 4

```
/* This program will compute the elements of the Concatenated Natural Sequence.
   Written by Charles Ashbacher. */

import java.math.*;
import java.io.*;

public class NaturalSequence
{
// The upper limit on the number of the element to be computed.
 static final int UPPERLIMIT=100;
```

```
// The base to be used in the computation.
 static BigInteger base1=new BigInteger("3");
 static String filename="Naturalvalues.txt";
 static File output;
 static BufferedWriter out;

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }

 public static void main(String[] args)
 {
  String str1;
  BigInteger theCount=new BigInteger("1");
  String base1String=new String("");
  BigInteger base1Value;
  long longEquivalent;
  String theConversion;
  System.out.println("The base is "+base1);
  try
  {
```

```
 output=new File(filename);
 output.createNewFile();
 if(!output.isFile())
 {
  System.out.println("Error creating the file");
  IOException ioe=new IOException();
  throw ioe;
 }
 out=new BufferedWriter(new FileWriter(output));
}
catch(IOException ioe)
{
 System.out.println("Cannot open the designated file");
 System.exit(0);
}

for(int i=1;i<UPPERLIMIT;i++)
{
 theConversion=ConvertToBase(theCount,base1);
 longEquivalent=theCount.longValue();
 base1String="";
 for(long counter=0;counter<longEquivalent;counter++)
 {
  base1String=base1String+theConversion;
 }
 base1Value=ConvertToDecimal(base1String,base1);
 System.out.println(base1String);
 System.out.println(base1Value);
 try
 {
  out.write("The base is "+base1);
  out.write(base1String);
 }
 catch(IOException ioe)
 {
  System.out.println("Could not write data to the file");
 }
 theCount=theCount.add(BigInteger.ONE);
}
System.out.println("The search was through "+UPPERLIMIT);

try
{
 out.write("The search was through "+UPPERLIMIT);
 out.close();
}
```

```
 catch(IOException ioe)
 {
  System.out.println("Could not close the file");
 }
 }
}
```

5) The Permutation Sequence

12, 1342, 135642, 13578642, 13579108642, 135791112108642, 1357911131412108642,

In general, the nth term of the sequence is created by concatenating the positive integers from 1 to 2n in the following way:

13 . . . (2n-3)(2n-1)(2n)(2n-2)(2n-4) . . . 42.

Clearly, since the last digit is an even number, none of the elements of this sequence can be prime. It is also possible to form the permutation sequence in other bases, and the program in listing five will compute all elements of a base B permutation sequence up to a set limit.

Listing 5

```
/* This program will create the elements of the Permutation Sequence for a preset base.
Written  by Charles Ashbacher. */

import java.math.*;
import java.io.*;

public class PermutationSequence
{
// Upper limit on the element number to compute.
 static final int UPPERLIMIT=10;
// The base used to construct the elements of the sequence.
 static BigInteger base1=new BigInteger("3");
 static String filename="Permutationvalues.txt";
 static File output;
 static BufferedWriter out;

/* Function to convert the BigInteger counter to the equivalent String form in the input
base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
```

```
 while(!theNumber.equals(BigInteger.ZERO))
 {
  theRemainder=theNumber.mod(theBase);
  theReturn=theRemainder.toString()+theReturn;
  theNumber=theNumber.divide(theBase);
 }
 return theReturn;
}

/* Function to convert the input String into the equivalent BigInteger in the input
BigInteger  base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
 BigInteger theReturn=new BigInteger("0");
 BigInteger thePositionValue;
 String theChar;
 int stringLength=theNumber.length();
 for(int i=0;i<stringLength;i++)
 {
  theChar=theNumber.substring(i,i+1);
  thePositionValue=new BigInteger(theChar);
  theReturn=theReturn.multiply(theBase).add(thePositionValue);
 }
 return theReturn;
}

 public static void main(String[] args)
 {
 BigInteger theCount=new BigInteger("1");
 String base1String=new String("");
 String base2String=new String("");
 String permutationString;
 BigInteger base1Value;
 String theConversion1;
 String theConversion2;
 System.out.println("The base is "+base1);
 try
 {
  output=new File(filename);
  output.createNewFile();
  if(!output.isFile())
  {
   System.out.println("Error creating the file");
   IOException ioe=new IOException();
```

```
  throw ioe;
 }
 out=new BufferedWriter(new FileWriter(output));
}
catch(IOException ioe)
{
 System.out.println("Cannot open the designated file");
 System.exit(0);
}
for(int i=1;i<=UPPERLIMIT;i++)
{
 theConversion1=ConvertToBase(theCount,base1);
 theConversion2=ConvertToBase(theCount.add(BigInteger.ONE),base1);
 base1String=base1String+theConversion1;
 base2String=theConversion2+base2String;
 permutationString=base1String+base2String;
 base1Value=ConvertToDecimal(permutationString,base1);
 System.out.println(permutationString);
 System.out.println(base1Value);
 try
 {
  out.write(permutationString);
  out.write(base1Value.toString());
 }
 catch(IOException ioe)
 {
  System.out.println("Could not write text to the file");
 }
 theCount=theCount.add(BigInteger.ONE).add(BigInteger.ONE);
}
System.out.println("The search was through "+UPPERLIMIT);

try
{
 out.write("The search was through "+UPPERLIMIT);
 out.close();
}
catch(IOException ioe)
{
 System.out.println("Could not close the file");
}
}
}
```

Since all of the elements of the sequence are even, none are prime. However, we can examine the sequence formed if we divide each of the elements of the Permutation Sequence by 2. The first elements of this sequence are

6, 671, 67821, 6789321, 6789554321, 67895556054321, 678955565706054321, 678955565657580706054321.

The program was modified to examine the first 400 elements of this sequence to determine which terms in this sequence were prime and two were found

13579111315171921232527282624222018161412108642 =

2 * 67895556575859606162636413121110090807060543 21

135791113151719212325272931333432302826242220181614121086 42 =

2 * 678955565758596061626364656667161514131211100908070605 4321

Notice that in both cases the last odd number concatenated to the element of the Permutation Sequence is evenly divisible by 3.

Also note that since each element of the Permutation Sequence has only one two as a factor, none can be a perfect power.

There are many distributed computing projects that are currently being run on the World Wide Web (WWW), and one of them is a search for primes of the form n! ± 1. Clearly, these numbers cannot be divided by any of the numbers 2 through n, so the probability that they are prime is higher than that of a randomly selected odd integer.

The elements of the Permutation Sequence are also built by combining all of the integers up through n, so a similar question that can be asked concerns the values of the Permutation Sequence plus and minus one.

[13 . . . (2n-3)(2n-1)(2n)(2n-2)(2n-4) . . . 42] + 1
[13 . . . (2n-3)(2n-1)(2n)(2n-2)(2n-4) . . . 42] – 1.

The program that computes the elements of the Permutation Sequence was then rerun up through element number 400 looking for primes having these forms. In the case where one was added, elements numbered 1, 9, 19, 61 and 701 were prime. When one was subtracted, only the first element 11, was prime.

Question: Are there an infinite number of primes of the form

13 . . . (2n-3)(2n-1)(2n)(2n-2)(2n-4) . . . 42 + 1?

Question: Are there any other primes in the sequence

[13 . . . (2n-3)(2n-1)(2n)(2n-2)(2n-4) . . . 42] – 1?

6) The Reverse Sequence

1, 21, 321, 4321, 54321, 654321, 7654321, 87654321, 987654321, 10987654321, 1110987654321, . . .

This sequence is easily defined as the nth element of the sequence is constructed by concatenating

n(n-1)(n-2) . . . 321

A program to compute the elements of the Reverse Sequence in an arbitrary base is given in listing 6.

Listing 6

```
/* This program will create the elements of the Reverse Sequence. Written by Charles Ashbacher. */

import java.math.*;
import java.io.*;

public class ReverseSequence
{
// Upper limit on the element number to compute.
 static final int UPPERLIMIT=10;
// The base used to construct the elements of the sequence.
 static BigInteger base1=new BigInteger("10");
 static String filename="Permutationvalues.txt";
 static File output;
 static BufferedWriter out;

/* Function to convert the BigInteger counter to the equivalent String form in the input base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
```

```
 }
 return theReturn;
}

/* Function to convert the input String into the equivalent BigInteger in the input
   BigInteger base. */

static BigInteger ConvertToDecimal(String theNumber,
                                   BigInteger theBase)
 {
 BigInteger theReturn=new BigInteger("0");
 BigInteger thePositionValue;
 String theChar;
 int stringLength=theNumber.length();
 for(int i=0;i<stringLength;i++)
 {
  theChar=theNumber.substring(i,i+1);
  thePositionValue=new BigInteger(theChar);
  theReturn=theReturn.multiply(theBase).add(thePositionValue);
 }
 return theReturn;
}

 public static void main(String[] args)
 {
 BigInteger theCount=new BigInteger("1");
 String base1String=new String("");
 BigInteger base1Value;
 String theConversion1;
 System.out.println("The base is "+base1);
 try
 {
  output=new File(filename);
  output.createNewFile();
  if(!output.isFile())
  {
   System.out.println("Error creating the file");
   IOException ioe=new IOException();
   throw ioe;
  }
  out=new BufferedWriter(new FileWriter(output));
 }
 catch(IOException ioe)
 {
  System.out.println("Cannot open the designated file");
  System.exit(0);
```

```
 }
 for(int i=1;i<=UPPERLIMIT;i++)
 {
  theConversion1=ConvertToBase(theCount,base1);
  base1String=theConversion1+base1String;
  base1Value=ConvertToDecimal(base1String,base1);
  if(base1Value.isProbablePrime(100))
  {
   try
   {
    System.out.println("Found a prime");
    System.out.println("Element number "+theCount);
    System.out.println(base1Value);
    out.write(base1String+"\n");
    out.write(base1Value.toString()+"\n");
   }
   catch(IOException ioe)
   {
    System.out.println("Could not write text to the file");
   }
  }
  theCount=theCount.add(BigInteger.ONE);
 }
 System.out.println("The search was through "+UPPERLIMIT);

 try
 {
  out.write("The search was through "+UPPERLIMIT);
  out.close();
 }
 catch(IOException ioe)
 {
  System.out.println("Could not close the file");
 }
}
}
```

This program was not run for a range of base 10 values. Others have examined all of the values in the Reverse Sequence have been examined through element number 10,000 and the only prime found was

828180 . . . 0987654321.

Some additional runs of the program were performed in order to examine some additional properties of the sequence. The following can be used to improve the search for primes and is very easy to prove.

Theorem: If n is the number of the element in the Reverse Sequence, then the element is evenly divisible by 3 if n is congruent to 0 or 2 modulo 3.

Proof: A simple modulus 3 argument.

Corollary: An element of the Reverse Sequence n(n-1) . . . 321 can be prime only when n is congruent to 1 modulo 3.

However, it is possible that there are more primes in the Reverse Sequence for bases other than 10, so the program was run to search for all primes in the first 500 elements of the Reverse Sequence for the bases 3 through 9. The results are summarized in table 4.

Table 4

Base	Elements that are prime
3	2, 5, 13, 57
4	4, 106, 118, 130
5	2, 313
6	2, 6, 17, 28, 33, 37, 81
7	373
8	2, 9, 47, 50, 99
9	2, 5, 346

The final sequence described by Smarandache that uses all the positive integers is the Antisymmetric Sequence.

7)The Antisymmetric Sequence

11, 1212, 123123, 12341234, 1234512345, 123456123456, 12345671234567, . . .

where element number n is defined by the concatenation

12..(n)12..(n)

Clearly, no element of this sequence, other than the first, can be prime, and that holds independent of the base. The program of listing 7 will compute the values of the Antisymmetric Sequence.

Listing 7

```
/* This program will compute the elements of the Antisymmetric Sequence.
   It was written by Charles Ashbacher. */

import java.math.*;
```

```
import java.io.*;

public class AntisymmetricSequence
{
// Upper limit of the number of elements to search.
 static final int UPPERLIMIT=100;
// The base to use in creating the elements of the sequence.
 static BigInteger base1=new BigInteger("3");
// The name of the file data can be sent to.
 static String filename="Antsymmetricvalues.txt";
 static File output;
 static BufferedWriter out;

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }

 public static void main(String[] args)
 {
  String str1;
  BigInteger theCount=new BigInteger("2");
```

```
String base1String=new String("1");
String totalString;
BigInteger base1Value;
String theConversion;
System.out.println("The base is "+base1);
try
{
 output=new File(filename);
 output.createNewFile();
 if(!output.isFile())
 {
  System.out.println("Error creating the file");
  IOException ioe=new IOException();
  throw ioe;
 }
 out=new BufferedWriter(new FileWriter(output));
}
catch(IOException ioe)
{
 System.out.println("Cannot open the designated file");
 System.exit(0);
}

 for(int i=1;i<UPPERLIMIT;i++)
 {
  theConversion=ConvertToBase(theCount,base1);
  base1String=base1String+theConversion;
  totalString=base1String+base1String;
  base1Value=ConvertToDecimal(totalString,base1);
//   System.out.println(theCount);
//   System.out.println(totalString);

 theCount=theCount.add(BigInteger.ONE);
}
System.out.println("The search was through "+UPPERLIMIT);

try
{
 out.write("The search was through "+UPPERLIMIT);
 out.close();
}
catch(IOException ioe)
{
 System.out.println("Could not close the file");
}
}
```

}

While searching for primes in the Antisymmetric Sequence is pointless, like the Permutation Sequence, we can ask if we can generate a prime when we add or subtract one from the element. The program that computes the elements of the Antisymmetric Sequence was modified to search for primes of the form

12..(n)12..(n) ± 1

up through element number 400.

Elements numbered 2 and 300 of the Antisymmetric Sequence plus 1 were identified as prime and element number 34 of the Antisymmetric Sequence minus 1 was identified as prime.

Question: Are there other primes in these sequences?

Examining the Divisibility Of Sequence Elements By Squares

Since the elements of the Permutation Sequence cannot be perfect powers, we can ask the same question about the elements of the Reverse Sequence. While this question is not so easy to answer, there is one property that is easy to prove.

Theorem: Element number n of the base 10 Reverse Sequence is not square-free if n is congruent to 0 or 8 modulo 9.

Proof: By induction on the modulus cycle, we prove that the element is divisible by 9 if n is congruent to 0 or 8 modulo 9.

Basis step

Element numbers 8 and 9 are

87654321 = 9 * 9739369 987654321 = 9 * 109739369.

Inductive step

Assume that for n = 9k + 8 and n + 1 = 9k + 9 the elements of the base 10 Reverse Sequence are evenly divisible by 9.

(9k+8)(9k+7) 321
(9k+9)(9k+8)(9k+7) . . . 321.

The number appended to the right of element 9k+8 to make element 9k + 9 + 8 would be

(9k+9+8)(9k+8+8)(9k+7+8)(9k+6+8)(9k+5+8)(9k+4+8)(9k+3+8)(9k+2+8)(9k+1+8).

Each of these numbers would be multiplied by a power of ten, all of which are congruent to 1 modulo 9. Therefore, if we express the numbers in their modulo 9 form

$(9k+9+8) * 10^k$ modulo 9 is 8*1 = 8
$(9k+8+8) * 10^k$ modulo 9 is 7*1 = 7
$(9k+7+8) * 10^k$ modulo 9 is 6*1 = 6
$(9k+6+8) * 10^k$ modulo 9 is 5*1 = 5
$(9k+5+8) * 10^k$ modulo 9 is 4*1 = 4
$(9k+4+8) * 10$ modulo 9 is 3*1 = 3
$(9k+3+8) * 10^k$ modulo 9 is 2*1 = 2
$(9k+2+8) * 10^k$ modulo 9 is 1*1 = 1
$(9k+1+8) * 10^k$ modulo 9 is 9*1 = 0

Therefore, the number

(9k + 9 + 8)(9k + 8 + 8)(9k + 7 + 8) . . .321

would have been formed by summing the 0 modulo 9 number

(9k+8)(9k+7) 321

and the sequence of numbers that are congruent to 0, 1, 2, 3, 4, 5, 6, 7 and 8 modulo 9 respectively. This sum is congruent to 0 modulo 9, so the overall sum is congruent to 0 modulo 9.

To verify that the next element of the Reverse Sequence is also congruent to 0 modulo 9, we note that we would append (9k+9) to the right of the previous element, which is known to be congruent to 0 modulo 9. Since

$(9k+9) * 10^k$

is also congruent to 0 modulo 9, element 9k + 9 + 9 must also be congruent to 0 modulo 9.

The program was then rerun searching for elements of the sequence that are divisible by 7^2 and the results for all elements with index less than or equal to 2000 are given in data listing 1.

Data listing 1

```
For term number 54, the number is divisible by 7 squared
The difference is 54
For term number 146,the number is divisible by 7 squared
The difference is 92
For term number 147, the number is divisible by 7 squared
The difference is 1
```

```
For term number 244, the number is divisible by 7 squared
The difference is 97
For term number 245, the number is divisible by 7 squared
The difference is 1
For term number 342, the number is divisible by 7 squared
The difference is 97
For term number 343, the number is divisible by 7 squared
The difference is 1
For term number 440, the number is divisible by 7 squared
The difference is 97
For term number 441, the number is divisible by 7 squared
The difference is 1
For term number 538, the number is divisible by 7 squared
The difference is 97
For term number 539, the number is divisible by 7 squared
The difference is 1
For term number 636, the number is divisible by 7 squared
The difference is 97
For term number 637, the number is divisible by 7 squared
The difference is 1
For term number 734, the number is divisible by 7 squared
The difference is 97
For term number 735, the number is divisible by 7 squared
The difference is 1
For term number 832, the number is divisible by 7 squared
The difference is 97
For term number 833, the number is divisible by 7 squared
The difference is 1
For term number 930, the number is divisible by 7 squared
The difference is 97
For term number 931, the number is divisible by 7 squared
The difference is 1
For term number 1043, the number is divisible by 7 squared
The difference is 112
For term number 1059, the number is divisible by 7 squared
The difference is 16
For term number 1084, the number is divisible by 7 squared
The difference is 25
For term number 1190, the number is divisible by 7 squared
The difference is 106
For term number 1206, the number is divisible by 7 squared
The difference is 16
For term number 1231, the number is divisible by 7 squared
The difference is 25
For term number 1337, the number is divisible by 7 squared
The difference is 106
For term number 1353, the number is divisible by 7 squared
```

```
The difference is 16
For term number 1378, the number is divisible by 7 squared
The difference is 25
For term number 1484, the number is divisible by 7 squared
The difference is 106
For term number 1500, the number is divisible by 7 squared
The difference is 16
For term number 1525, the number is divisible by 7 squared
The difference is 25
For term number 1631, the number is divisible by 7 squared
The difference is 106
For term number 1647, the number is divisible by 7 squared
The difference is 16
For term number 1672, the number is divisible by 7 squared
The difference is 25
For term number 1778, the number is divisible by 7 squared
The difference is 106
For term number 1794, the number is divisible by 7 squared
The difference is 16
For term number 1819, the number is divisible by 7 squared
The difference is 25
For term number 1925, the number is divisible by 7 squared
The difference is 106
For term number 1941, the number is divisible by 7 squared
The difference is 16
For term number 1966, the number is divisible by 7 squared
The difference is 25
```

From this list, note that for elements numbered between 100 and 1000, the sequence of differences between elements divisible by 17^2 follow the 1, 97 cycle, and for the elements numbered greater than 1000, the difference cycle is 16, 25, 106. This certainly raises the question of what the cycle is for elements numbered greater than 10,000.

To investigate this question, the program was rewritten to determine what the cycle is for elements numbered greater than 10,000. That cycle turned out to be 82, 29, 58, 22, 51, 52. While the change in the cycles is understandable, given that numbers with an extra digit are being appended, there is no obvious explanation for the values of the cycle.

Question: Are there any divisibility properties that can be used to predict the numbers in the division by 7^2 cycle for elements numbered greater than 100,000?

The program was run through the first 1,200 terms searching for the elements of the Reverse Sequence that are evenly divisible by 11^2, and the results appear in data listing 2.

Data listing 2

```
For term number 18, the number is divisible by 11 squared
```

```
The difference is 18
For term number 36, the number is divisible by 11 squared
The difference is 18
For term number 219, the number is divisible by 11 squared
The difference is 183
For term number 264, the number is divisible by 11 squared
The difference is 45
For term number 461, the number is divisible by 11 squared
The difference is 197
For term number 506, the number is divisible by 11 squared
The difference is 45
For term number 703, the number is divisible by 11 squared
The difference is 197
For term number 748, the number is divisible by 11 squared
The difference is 45
For term number 945, the number is divisible by 11 squared
The difference is 197
For term number 990, the number is divisible by 11 squared
The difference is 45
For term number 1008, the number is divisible by 11 squared
The difference is 18
For term number 1037, the number is divisible by 11 squared
The difference is 29
For term number 1129, the number is divisible by 11 squared
The difference is 92
For term number 1158, the number is divisible by 11 squared
The difference is 29
For term number 1250, the number is divisible by 11 squared
The difference is 92
For term number 1279, the number is divisible by 11 squared
The difference is 29
For term number 1371, the number is divisible by 11 squared
The difference is 92
For term number 1400, the number is divisible by 11 squared
The difference is 29
For term number 1492, the number is divisible by 11 squared
The difference is 92
For term number 1521, the number is divisible by 11 squared
The difference is 29
For term number 1613, the number is divisible by 11 squared
The difference is 92
For term number 1642, the number is divisible by 11 squared
The difference is 29
For term number 1734, the number is divisible by 11 squared
The difference is 92
For term number 1763, the number is divisible by 11 squared
The difference is 29
```

```
For term number 1855, the number is divisible by 11 squared
The difference is 92
For term number 1884, the number is divisible by 11 squared
The difference is 29
For term number 1976, the number is divisible by 11 squared
The difference is 92
The search was through 2000
```

For divisibility by 11^2, note that in the interval 100 to 1000, the difference cycle is 197, 45 and for greater than 1000, the difference cycle is 29, 92. This also raises the obvious question of what the cycle is for elements numbered greater than 10,000. The program was modified and run again to examine what the cycle is for elements numbered greater than 10,000 and the cycle is 89, 153.

As was the case for divisibility by 7^2, there is no obvious reason for these values. Therefore, the same question can be answered.

Question: Are there any divisibility properties that can be used to predict the numbers in the division by 11^2 cycle for elements numbered greater than 100,000?

Chapter 2
Sequences Formed By Concatenating Selected Natural Numbers

F. Smarandache defined a series of sequences formed by concatenating selected natural numbers. In this chapter, those sequences will be listed, along with Java programs that will compute the elements of the sequences. Some properties of these sequences will also be proven.

1) Concatenated Odd Sequence

1, 13, 135, 1357, 13579, 1357911, 135791113, . . .

For any integer $n \geq 1$, the nth term of the sequence is formed by concatenating all the odd numbers 1 through $2n - 1$ left to right.

Smarandache conjectured that there are infinitely many primes in this sequence.

The search for primes can be made easier by incorporating the following properties of the sequence.

Theorem:
Let n be the number of the element in the Concatenated Odd sequence.

a) If n is congruent to 3 modulo 5, then element number n is evenly divisible by 5.
b) If n is congruent to 0 modulo 3, then element number n is evenly divisible by 3 and if n is congruent to 1 or 2 modulo 3, then element number n is congruent to 1 modulo 3.

Proof:

a) Element number 3 is 135, so it is evenly divisible by 5.

Assume that for n congruent to 3 modulo 5, the trailing digit is 5. The next five elements concatenated on the right will have trailing digits 7, 9, 1, 3 and 5 respectively. Since number of the last element of this list will also be congruent to 3 modulo 5 and the trailing digit of the number will be 5, it too will be divisible by 5.

b) We start this proof with a lemma,

Lemma: For the Concatenated Odd Sequence, if the index is congruent to 1 modulo 3, the last number concatenated on the right is congruent to 1 modulo 3. If the index is congruent to 2 modulo 3, then the last number concatenated on the right is congruent to 0 modulo 3 and if the index is congruent to 0 modulo 3, the last number concatenated on the right is congruent to 2 modulo 3.

Proof: By induction on the index.

Start with the first three numbers of the sequence, 1, 13, and 135, where the indices are congruent to 1, 2 and 0 modulo three and the last numbers concatenated on the right are congruent to 1, 0 and 2 modulo three respectively.

Let n be an index congruent to 1 modulo 3 and assume that the last number concatenated on the right was congruent to 1 modulo 3. The index of the next element will of course be congruent to 2 modulo three and since we added 2 to the previous number concatenated on the right, the last number on the right of the next element will be congruent to 0 modulo 3. The index of the next element will be congruent to 0 modulo 3 and adding 2 to the previous number concatenated on the right, the last number on the right of this element will be congruent to 2 modulo 3. Doing this one more time will yield an element whose index is congruent to 1 modulo 3 and where the last number concatenated on the right is congruent to 1 modulo 3. This starts the cycle over again.

Proof of part (b).

Element numbers one, two and three are 1, 13 and 135, and the first two are congruent to 1 modulo 3 and the last is evenly divisible by 3.

Assume that for n = 3k +1 and n+1 = 3k + 2 the Concatenated Odd Sequence elements are congruent to 1 modulo 3 and for n + 2 = 3k + 3 the element is congruent to 0 modulo 3.

Let $d_k \ldots d_1$ represent element numbered n + 2, which is evenly divisible by 3. Form element number n + 3 by concatenating the next element K, where K is congruent to 1 modulo 3. This number can be represented as

$(d_k \ldots d_1) * 10^r + K$

Since the product is evenly divisible by 3 and K is congruent to 1 modulo 3, the sum is congruent to 1 modulo 3.

Form the next element by concatenating K + 1 to the previous one. This number can be represented in the form

$[(d_k \ldots d_1) * 10^r + K]10^s + (K+1)$

Since $(d_k \ldots d_1) * 10^r$ was congruent to 1 modulo 3 and multiplication by a power of ten does not change that and from the lemma, K + 1 is congruent to 0 modulo 3. Therefore, the sum is congruent to 1 modulo 3.

Forming the next element by concatenating K + 2 to the previous term, it would have the form

$([(d_k \ldots d_1) * 10^r + K]10^s + (K+1)) * 10^t + (K + 2)$

since the previous term was congruent to 1 modulo 3, the product will also be congruent to 1 modulo 3. From the lemma, K + 2 is congruent to 2 modulo 3, so the sum is congruent to 0 modulo 3.

The next element is formed by concatenating K + 3 to the right, which starts the cycle over again.

These properties of the elements of the sequence allow for some filtering to be done when we search it for primes.

A program to compute the elements of this sequence is given in listing 1.

Listing 1

```
/* This program will create the elements of the Concatenated Odd Sequence and check to
   see if they are prime. Written by Charles Ashbacher. */

import java.math.*;
import java.io.*;

public class ConcatenatedOddSequence
{
// Upper limit on the element number to compute.
 static final int UPPERLIMIT=300;
 static final BigInteger theTHREE=new BigInteger("3");
// The base used to construct the elements of the sequence.
 static BigInteger base1=new BigInteger("10");
 static String filename="OddSequencevalues.txt";
 static File output;
 static BufferedWriter out;

/* Function to convert the BigInteger counter to the equivalent String form in the input
base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }
```

```
/* Function to convert the input String into the equivalent BigInteger in the input
   BigInteger  base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }

 public static void main(String[] args)
 {
  BigInteger theCount=new BigInteger("1");
  String base1String=new String("");
  BigInteger base1Value;
  String theConversion1;
  String str1;
  int modthree=1;
  System.out.println("The base is "+base1);
  try
  {
   output=new File(filename);
   output.createNewFile();
   if(!output.isFile())
   {
    System.out.println("Error creating the file");
    IOException ioe=new IOException();
    throw ioe;
   }
   out=new BufferedWriter(new FileWriter(output));
  }
  catch(IOException ioe)
  {
   System.out.println("Cannot open the designated file");
   System.exit(0);
  }
```

```
 for(int i=1;i<=UPPERLIMIT;i++)
 {
  theConversion1=ConvertToBase(theCount,base1);
  base1String=base1String+theConversion1;
  base1Value=ConvertToDecimal(base1String,base1);
  if (base1Value.isProbablePrime(100))
  {
   try
   {
    str1="Last number concatenated "+theCount+"\n";
    str1=str1+"Is prime\n";
    str1=str1+base1String+"\n";
    str1=str1+base1Value+"\n";
    System.out.println(str1);
    out.write(str1);
   }
   catch(IOException ioe)
   {
    System.out.println("Could not write text to the file");
   }
  }
  theCount=theCount.add(BigInteger.ONE).add(BigInteger.ONE);
 }
 System.out.println("The search was through "+UPPERLIMIT);

 try
 {
  out.write("The search was through "+UPPERLIMIT);
  out.close();
 }
 catch(IOException ioe)
 {
  System.out.println("Could not close the file");
 }
}
}
```

This program was run in a search for prime numbers, and elements numbered 2, 10, 16, 34 and 49 were found to be prime.

While Smarandache did not mention the creation of similar Concatenated Odd sequences in other bases in the follow up to the original definition, it was mentioned in some other sequences, so we will consider them here. For example, the first few elements of the Concatenated Odd Sequence for the base 3 are

1, 110, 11012, 1101221, 1101221100, 1101221100102, 1101221100102111, . . .

The previous program was modified so that it can be used to create the elements of the Concatenated Odd Sequence for different bases. It was run for the first 300 elements of all sequences with the base equal to 3 through 9 and the results are summarized in table 1.

Table 1

Base	**Index values that are prime**
3	3, 8, 13
4	2, 4, 11, 14, 64, 74
5	None
6	3, 4, 19, 67
7	17, 155
8	2, 4, 5, 25
9	3

Note that the third and fourth elements of three of the sequences are prime and that no primes were found when the base was five. Also note that the number of primes is larger for even bases than for the odd. These results lead to the following questions.

Question: Is there an element in the Concatenated Odd Sequence base five sequence which is prime?

Question: Are there more primes in the Concatenated Odd Sequences for even bases than there are in the Concatenated Odd Sequences for odd bases?

To examine the first question in more detail, the program was rerun to examine the first 500 elements in the base five sequence and no prime was found.

2) Concatenated Even Sequence

2, 24, 246, 2468, 246810, 24681012, 2468101214, . . .

Obviously, with the exception of 2, none of these numbers can be prime. Therefore, Smarandache posed the question:

Question: How many elements of the Concatenated Even Sequence are a perfect power?

He then conjectured that the answer was no.

It is a simple matter to modify the program that creates the elements of the Concatenated Odd Sequence so that it creates the elements of the Concatenated Even Sequence. Simply change the line

```
BigInteger theCount=new BigInteger("1");
```

to

```
BigInteger theCount=new BigInteger("2");
```

While searching the sequence for primes is not an option, it is possible that elements can be twice a prime. Therefore, the program to create the elements of the Concatenated Odd Sequence was modified to search for members that are twice a prime.

The sequence created by dividing the elements of the Concatenated Even Sequence by 2 has an interesting appearance and the first few terms are

1, 12, 123, 1234, 123405, 12340506, 1234050607, 123405060708, 12340506070809, 1234050607080910, 123405060708091011, . . .

The program was run for the first 500 terms in the sequence and the only number that is twice a prime was

2468101214 = 2 * 1234050607

While the second element of the sequence, 12 is not twice a prime, it is 4 times a prime, or more generally the product of a power of two and an odd prime. Therefore, a more general question that can be asked is, "How many elements of the Concatenated Even Sequence are the product of an odd prime and a power of two?" The program was then modified to search for solutions to this question and examined the first 300 elements of the sequence. Only three solutions were found and they are

24 = 3 * 4, 2468 = 617 * 4, 2468101214 = 2 * 1234050607

The properties of the elements of the Concatenated Even Sequence when divided by two leads to a simple theorem concerning whether elements can be a perfect power.

Theorem: An element of the Concatenated Even Sequence can be a perfect power only when the last number concatenated on the end is divisible by 4.

Proof: It is easy to verify that the elements of the Concatenated Even Sequence divided by two alternate between even and odd. If there is only one factor of two in the element of the Concatenated Even Sequence, then it cannot be a perfect power. If the last number concatenated on the right is divisible by 4, then there is more than one factor of two in the element.

Corollary: Elements of odd index in the Concatenated Even Sequence cannot be perfect squares.

Proof: If k is the index of the element, then the last number concatenated on the rights was 2k. This is not divisible by 4 if k is odd.

Additional information concerning the Concatenated Even Sequence can be gleaned by examining the patterns of divisibility by three. In examining the first elements of the sequence, it is immediately noted that the elements numbered 2, 3, 5, 6, 8, 9, 11 and 12 are all evenly divisible by 3. These numbers are all either congruent to 0 or 2 modulo 3 and it is easy to prove that this is the general behavior.

Theorem: If k is the index of element n of the Concatenated Even Sequence, then if k is congruent to 0 or 2 modulo 3, n is divisible by 3.

Proof: The proof is by induction.

Basis step: For k = 2, n = 24 and for k = 3, n = 246, both evenly divisible by 3.

Inductive step: Assume that for index k, where k is congruent to 0 modulo 3, element number k , call it n, is evenly divisible by 3. Concatenate 2(k+1), 2(k+2) and 2(k+3) to the right of n to form element numbered k+3

$$n(2(k+1))(2(k+2))(2(k+3)).$$

The sums of the digits of the three numbers concatenated on the right would be some combination of 0, 1 and 2 modulo 3. The sum of these sums would be evenly divisible by 3, so element k+3 would also be evenly divisible by 3.

The proof for k congruent to 2 modulo 3 is almost identical.

As a next step, we note that elements numbered 8, 9, 17, 18, 26 and 27 are all evenly divisible by 9. This is again a general behavior and not difficult to prove, so the proof is omitted.

Theorem: If n is an element of the Concatenated Even Sequence, then n is evenly divisible by 9 only when the index of the element is congruent to 0 or 8 modulo 9.

Since all other elements that are divisible by 3 have only one 3 as a factor, they cannot be perfect powers. Therefore, if we combine the theorem on odd indices with the previous one, we have the following result.

Theorem: For an element of the Concatenated Even Sequence to be a perfect power, the index k must be even and it must be congruent to 1 modulo 3 unless it is congruent to 0 or 8 modulo 9.

Using the next smallest prime 5, we can perform some additional filtering of elements of the Concatenated Even Sequence that cannot be perfect powers.

Theorem: If k is the index of an element of the Concatenated Even Sequence, then k cannot be a perfect power if k is congruent to 0 modulo 10 unless k is divisible by 25.

Proof: If k has the form 10j, then the last value concatenated on the right was 20j, which will be divisible by 5 only one time, unless j is also divisible by 5. Since the segment of the sequence element to the left of this is divisible by 5, the entire number will have only one five factor if 20j does. With only one instance of 5 in the element, it cannot be a perfect power.

3) Smarandache Concatenated Prime Sequence

2, 23, 235, 2357, 235711, 23571113, 2357111317, . . .

It is conjectured that there are an infinite number of primes in this sequence.

The first 1, 624 terms have been examined and terms numbered 1, 2, 4, 128, 174, 342, 435 and 1429 were found to be prime. However, to date no one has settled the conjecture.

The arguments that could possibly be used to prove that the prime sequence contains an infinite number of primes may be found by examining the behavior of similar sequences. For example, we can determine the number of primes in the sequence formed by the odd primes

3, 35, 357, 35711, 3571113, 357111317, . . .

The program to search the Prime Sequence for primes was modified to search for primes in the first 1000 terms of the above sequence. With the exception of the first element, no primes were found.

Question: Is there an element in the sequence 3, 35, 357, 35711, . . . other than the first that is prime?

To some people, 1 is also a prime, so we can ask the similar question regarding the number of primes in the sequence

1, 12, 123, 1235, 12357, 1235711, 123571113, . . .

The program to search the Prime Sequence for primes was then modified to search for primes in the first 600 terms in the above sequence. Other than the initial term, only the terms numbered 6 (1235711) and 10 (123571113171923) were prime. When this is contrasted with the number of primes in the Prime Sequence (elements numbered 1, 2, 4, 128, 174, 342, 435 and 1429 are prime), the initial evidence is that there is something different about the Prime sequence.

The program was modified to search the terms of the sequences and count the number of terms that are congruent to zero, one and two modulo three. It was run for the first 200 terms in each of the three sequences, and the results are summarized in table 2.

Table 2

Sequence	# 0 mod 3	# 1 mod 3	# 2 mod 3
1, 12, 123, 1235, …	76	57	67
2, 23, 235, 2357, …	57	67	76
3, 35, 357, 35711, …	76	57	67

The main point of the table is to note that the numbers of values congruent to 0, 1 and 2 modulo three in each of the sequences are roughly the same. The fact that they are the same across the sequences (with offset) is of course no surprise. The terms of the first two sequences can be matched in the following way,

12 -> 2 123 -> 23 1235 -> 235 12357 -> 2357

so the terms differ by one, with the top being the larger. Therefore, the numbers for the top are rotated one entry to the right.

The terms of the second and third can be matched in the following way.

23 -> 3 235 -> 35 2357 -> 357 235711 -> 35711

so there is a difference of two and the rotation of the numbers reflects that.

In this sample, the number of elements that are congruent to 0 modulo 3 is larger for the sequences that begin with 1 and 3 than it is for the sequence that begins with 2. Which brings us to the question,

What are the true percentages of elements in these three sequences that are congruent to 0 modulo 3?

To further test this, the program was rerun to examine the first 1000 elements in the sequence 3, 35, 357, . . . and the number of elements in the sequence that were 0, 1 and 2 modulo 3 were 339, 342 and 319 respectively. From these numbers, it appears that the percentages are near equality. This is certainly reasonable and is in conformance with other theories regarding prime numbers. Therefore, there is no reason to believe that the differences in the numbers of primes in these sequences has anything to do with divisibility by three.

The similar sequence consisting of one and the odd primes concatenated

1, 13, 135, 1357, 135711, 13571113, 1357111317, 135711131719, ...

was also examined through the first 300 terms. Two primes were found, 13 and 13571113. Contrast this with the fact that no primes were found in the sequence beginning with 3. The numbers of elements congruent to zero, one and two modulo three were also roughly equal.

The situation is quite different for the elements of the sequence that starts with two ones

1, 11, 113, 1135, 11357, 1135711, 113571113, 11357111317, 1135711131719, . . .

When the first 300 terms of this sequence are examined, ten primes are found and the element indices are 2, 3, 6, 8, 9, 15, 28, 57, 102 and 146. The relatively large number of elements that are prime raises interesting questions about what is different between this sequence and the others.

To continue the examination of these sequences, the program was rewritten to examine the values of the terms modulo 7. It was used to count the numbers of elements of each of the three sequences

1, 12, 123, 1235, 12357, 1235711, . . .
2, 23, 235, 2357, 235711, 23571113, . . .
3, 35, 357, 35711, 3571113, 357111317, . . .

that were congruent to 0, 1, 2, 3, 4, 5, and 6 modulo 7. Examining the first 300 elements of each sequence, there was no significant difference between the numbers. The lowest of the counts was 37 and the highest was 53.

As the last part of the test, the program was modified to count the elements of these three sequences that are congruent to 0 through 10 modulo 11. It was run for the first 300 terms of all three sequences and the numbers of elements congruent to each of the numbers 0 through 10 modulo 11 differ somewhat, but not substantially so. This leads to another question that can be asked concerning these sequences.

Problem: Is there a prime number p such that the relative values of the possible remainders of the elements of these sequences modulo p are significantly different?

The larger number of primes in the sequence 1, 11, 113, 1135, . . . is a hint that perhaps there is something different regarding the distribution of the counts modulo the small primes. However, when the program was modified to determine the remainder counts modulo the small primes for this sequence, no significant difference from those of the other sequences was discovered.

The numbers of the elements of the sequences that are congruent to the various values of the remainders modulo a prime is related to the problem known as the Prime Number Race. Let $\pi(n;a,b)$ represent the number of primes $p \leq n$, that are congruent to a modulus b. The race, a particular interest of Pál Turán, is to determine the relative values of

л(n;a_1,b) and л(n;a_2,b) for a_1 not congruent to a_2 modulo b. In particular, Chebyshev noted that л(n;1,3) < л(n;2,3) and л(n;1,4) < л(n;3,4) for small value of n[1].

The relative values of the number of primes modulo a specific number could possibly be used to explain some of these results. Since we are concatenating the primes, the affect would not be direct. However, if there are an unusual number of primes with a specific modulo value, the continued concatenation of primes with that value could possibly skew the results and lead to the disparity in the number of primes in the sequences.

To examine this further, the program was modified to determine the counts of primes congruent to 1 modulo 3 and congruent to 2 modulo 3. The program was run for the first 400 odd primes, then for the first 1000 and finally for the first 500. While it was true that in all cases, л(n;1,3) < л(n;2,3), the percentage differences were quite small and these differences are summarized in table 3.

The process was repeated for the counts of primes congruent to 1 and 3 modulo 4 and the results are summarized in table 4.

Table 3

Number of odd primes	л(n;1,3)	л(n;2,3)	% л(n;2,3)
400	196	204	51
1000	491	508	50.8
5000	2484	2515	50.3

Table 4

Number of odd primes	л(n;1,4)	л(n;3,4)	% л(n;3,4)
400	196	204	51
1000	495	505	50.5
5000	2486	2514	50.28

The results summarized in tables 3 and 4 reinforce all other results, namely that the values of the primes and the concatenation of the primes modulo numbers, prime or not prime, appear to be randomly distributed.

Once the issue of the apparent appearance of random behavior is raised, we can ask companion questions concerning the relationship between the elements of this sequence and the primes.

Question: How many of the elements of the Concatenated Prime Sequence are two less than a prime?

Question: How many of the elements of the Concatenated Prime Sequence are two more than a prime?

The program was modified to search for prime numbers that are two less or two more than an element of the Concatenated Prime Sequence. Elements numbered 3, 11 and 24 minus 2 and element number 164 plus 2 are the only primes in the first 500 elements of the sequence. The number of primes found is consistent with the idea that the elements of the Concatenated Prime Sequence are randomly distributed around the prime numbers.

4) Smarandache Concatenated Square Sequence

1, 14, 149, 14916, 1491625, ...

where element number k is formed by concatenating the repeated squares on the right.

1(2*2)(3*3) . . . (k*k)

The question that was then posed was:

How many elements of this sequence are perfect squares?

A program was written to examine the elements of this sequence. For every element of the Concatenated Square sequence m, it uses a binary search method to determine the largest number k, such that $k*k \leq m$. If $k*k = m$, which means the element of the sequence is a perfect square, that fact is noted. The program was run for the first 1000 elements of the sequence and other than the first term, no perfect squares were found.

The program appears in listing 2.

Listing 2

```
/* This program will compute the values of the elements of the Smarandache
   Concatenated Square sequence. It will determine the smallest perfect square (k*k) that
   is less than or equal to the element and then compute the difference

         SmarandacheConcatenatedSquareElement - k*k

   The differences will then be written to a file for plotting. Written by Charles
    Ashbacher. */

import java.math.*;
import java.io.*;

public class ConcatenatedSquare
{
// The number of elements to compute.
```

```
 static final int UPPERLIMIT=500;
 static BigInteger base1=new BigInteger("10");
 static final BigInteger theTWO=new BigInteger("2");
 static final BigInteger theNINE=new BigInteger("9");
 static final BigInteger theTEN=new BigInteger("10");
 static final BigInteger theHUNDRED=new BigInteger("100");

// File where data is written.
 static String filename="ConcatenatedSquares.txt";
 static File output;
 static BufferedWriter out;

/* Function to convert the BigInteger counter to the equivalent String form in the input
base. */

 static String ConvertToBase(BigInteger theNumber,BigInteger theBase)
 {
  BigInteger theRemainder;
  String theReturn=new String("");
  while(!theNumber.equals(BigInteger.ZERO))
  {
   theRemainder=theNumber.mod(theBase);
   theReturn=theRemainder.toString()+theReturn;
   theNumber=theNumber.divide(theBase);
  }
  return theReturn;
 }

/* Function to convert the input String into the equivalent BigInteger in the input
    BigInteger base. */

 static BigInteger ConvertToDecimal(String theNumber,
                                    BigInteger theBase)
 {
  BigInteger theReturn=new BigInteger("0");
  BigInteger thePositionValue;
  String theChar;
  int stringLength=theNumber.length();
  for(int i=0;i<stringLength;i++)
  {
   theChar=theNumber.substring(i,i+1);
   thePositionValue=new BigInteger(theChar);
   theReturn=theReturn.multiply(theBase).add(thePositionValue);
  }
  return theReturn;
 }
```

```
/* This function accepts a BigInteger in String form and determines the largest
   perfect square that is less than or equal to the input. It returns the square
   root of the largest perfect square. */

 static BigInteger GetIntegerSquareRoot(String theElement)
 {
// Convert the input string into a BigInteger
  BigInteger theValue=ConvertToDecimal(theElement,base1);
  BigInteger lowValue,highValue,middleValue;
  BigInteger testValue;
  BigInteger lastHighDigit;
  int numberDigitsInNumber=theElement.length();
  int i,valueAsInt,rootAsInt;
  boolean quittest;
  int numberDigitsInRoot;
  int compareTest;
// First digits of the lower and upper bounds of the initial range containing the
// square root.
  Integer lowLeadingDigits,highLeadingDigits;
  int sizeOfLeadingDigitFlag=numberDigitsInNumber%2;

// If the input is less than one hundred, simply use the library functionality to
// determine the square root.
  compareTest=theValue.compareTo(theHUNDRED);
  if(compareTest<0)
   {
   valueAsInt=theValue.intValue();
   rootAsInt=(int)Math.sqrt(valueAsInt);
   Integer smallResult=new Integer(rootAsInt);
   middleValue=new BigInteger(smallResult.toString());
  }
// The input has at least three digits.
  else
   {
// If the number of digits in the input is even, then the square root is somewhere between
// 310 . . .0 and 999 . . . 99 where the number of digits is half the number of digits in
// the input.
   if((sizeOfLeadingDigitFlag)==0)
   {
    numberDigitsInRoot=numberDigitsInNumber/2;
    lowLeadingDigits=new Integer(31);
    highLeadingDigits=new Integer(99);
    lastHighDigit=theNINE;
   }
   else
```

```
// If the number of digits in the input is odd, then the square root is somewhere between
// 10 . . .0 and 320 . . .0, where the number of digits is half the number of digits in the
// input plus one.
   {
    numberDigitsInRoot=(numberDigitsInNumber+1)/2;
    lowLeadingDigits=new Integer(10);
    highLeadingDigits=new Integer(32);
    lastHighDigit=BigInteger.ZERO;
   }

// Initialize the values of the lowest possible value and the highest possible value
   lowValue=new BigInteger(lowLeadingDigits.toString());
   for(i=0;i<numberDigitsInRoot-2;i++)
   {
    lowValue=lowValue.multiply(theTEN);
   }

   highValue=new BigInteger(highLeadingDigits.toString());
   for(i=0;i<numberDigitsInRoot-2;i++)
   {
    highValue=highValue.multiply(theTEN).add(lastHighDigit);
   }

// Determine the value halfway between the two range values.
   middleValue=lowValue.add(highValue).divide(theTWO);
   compareTest=middleValue.compareTo(lowValue);
   quittest=false;

/* The search algorithm is the basic binary root finding algorithm. We start with the low
    and high values of the range and then compute the midpoint. If the value we are
    looking for is less than the midpoint, we set the high value to the midpoint, otherwise
    we set the low value to the midpoint. If the midpoint is what we are looking for, we
    make an assignment and immediately exit the loop. The loop will end when either we
    find a solution or when the  middleValue is the same as the lower bound. */

   while((compareTest>0)&&(!quittest))
   {
    testValue=middleValue.multiply(middleValue);
    compareTest=theValue.compareTo(testValue);
    if(compareTest>0)
    {
     lowValue=middleValue;
    }
    else
    {
     if(compareTest<0)
```

```
    {
     highValue=middleValue;
    }
    else
    {
     quittest=true;
    }
   }
   middleValue=lowValue.add(highValue).divide(theTWO);
   compareTest=middleValue.compareTo(lowValue);
  }
 }
 return middleValue;
}

 public static void main(String[] args)
 {
// This allows the evaluation to be restarted at any element in the Concatenated Square
// sequence.
  int startElementCount=2;
  Integer initialInt=new Integer(startElementCount);
// This will store the number of the element in the sequence.
  BigInteger elementCount=new BigInteger(initialInt.toString());
  BigInteger theRootSquared;
  String base1String=new String("");
  String differenceString;
  BigInteger base1Value;
  BigInteger theCount1;
  BigInteger theRoot;
  BigInteger theDifference;
  String theConversion1;
  String str1;
  int squareTest;
  int digitsInDifference;
  try
  {
   output=new File(filename);
   output.createNewFile();
   if(!output.isFile())
   {
    System.out.println("Error creating the file");
    IOException ioe=new IOException();
    throw ioe;
   }
   out=new BufferedWriter(new FileWriter(output));
  }
```

```
  catch(IOException ioe)
  {
   System.out.println("Cannot open the designated file");
   System.exit(0);
  }

// Initialize the string to the element of the sequence the evaluation is to start at
  for(int i=1;i<startElementCount;i++)
  {
   base1String=base1String+(i*i);
  }

  for(int i=1;i<=UPPERLIMIT;i++)
  {
   System.out.println("elementCount "+elementCount);

// Square the current element number.
   theCount1=elementCount.multiply(elementCount);
// Used when another base is to be used. Unnecessary if the base is 10.
   theConversion1=ConvertToBase(theCount1,base1);
   base1String=base1String+theConversion1;

// Convert the string into the equivalent number in the base base1.
   base1Value=ConvertToDecimal(base1String,base1);

// Find the largest number k, such that k*k is less than or equal to the current element
// of the Concatenated Square sequence.
   theRoot=GetIntegerSquareRoot(base1String);

// Square the root and compare it to the element of the Concatenated Square sequence
// If it is equal, print a message and stop processing.
   theRootSquared=theRoot.multiply(theRoot);
   squareTest=base1Value.compareTo(theRootSquared);
   if(squareTest==0)
   {
    System.out.println("We have a perfect square");
    break;
   }

// Compute the difference between the element and the square of the "root", then
// convert it into a String and determine the number of elements in the String.
// This will be the number of digits in the difference.
   theDifference=base1Value.subtract(theRootSquared);
   differenceString=theDifference.toString();
   digitsInDifference=differenceString.length();
```

```
// Write everything to the file.
   try
    {
     out.write(elementCount.toString()+","+digitsInDifference+"\n");
    }
    catch(IOException ioe)
    {
     System.out.println("Could not write text to the file");
    }
   elementCount=elementCount.add(BigInteger.ONE);
  }
  try
  {
   out.write("The search was through "+UPPERLIMIT);
   out.close();
  }
  catch(IOException ioe)
  {
   System.out.println("Could not close the file");
  }
 }
}
```

The program was then modified to compute the differences between k*k and the element of the sequence m. For example, the differences for the first six elements (after the first) are summarized in table 5.

Table 5

Element number	Value	k	Value – k*k
2	14	3	5
3	149	12	5
4	14916	122	32
5	1491625	1221	784
6	149162536	12213	5167
7	14916253649	1221321	379923

From this table, there are clear indications that the differences are increasing. To further test this, the program was modified to run through the first 500 elements of the Concatenated Square sequence and compute the differences. Because the numbers get large very quickly, the number of digits in the difference is what is plotted on the y-axis, rather than the value of the difference. This data appears in figure 1.

Since the number of digits is what is plotted on the y-axis, the plot is roughly n versus $\log_{10}n$. Furthermore, the shape of the plotted graph shows a relationship that grows faster than a simple linear one.

However, while this gives us some hints as to the lower bound behavior, it tells us nothing about the rate of change of the differences between the smallest perfect square greater than or equal to the element of the Concatenated Square sequence and the element. It is conceivable that those differences are shrinking. Therefore, the program was rewritten to examine the differences on the upper end, and then run for the first 500 elements of the sequence. The results appear in figure 2. Once again, the differences increase and exhibit a log to the base 10 behavior.

The combination of these two results make the following conjecture, made by Florentin Smarandache, a safe one.

Conjecture: No element of the Concatenated Square Sequence is a perfect square.

There is another question that can be asked that is more fundamental and where an affirmative answer would provide a solution to the conjecture.

Definition: Define the function Diffsquare(n) on the natural numbers to be the difference between the nth element of the Concatenated Square sequence and the largest perfect square k*k that is less than or equal to the nth element.

Question: Is Diffsquare(n) an always increasing function?

Figure 1

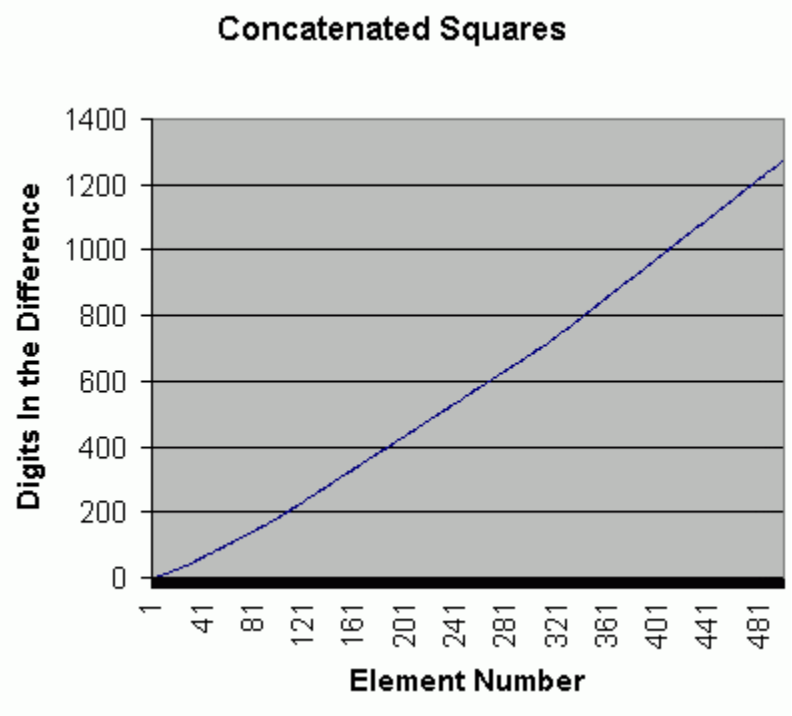
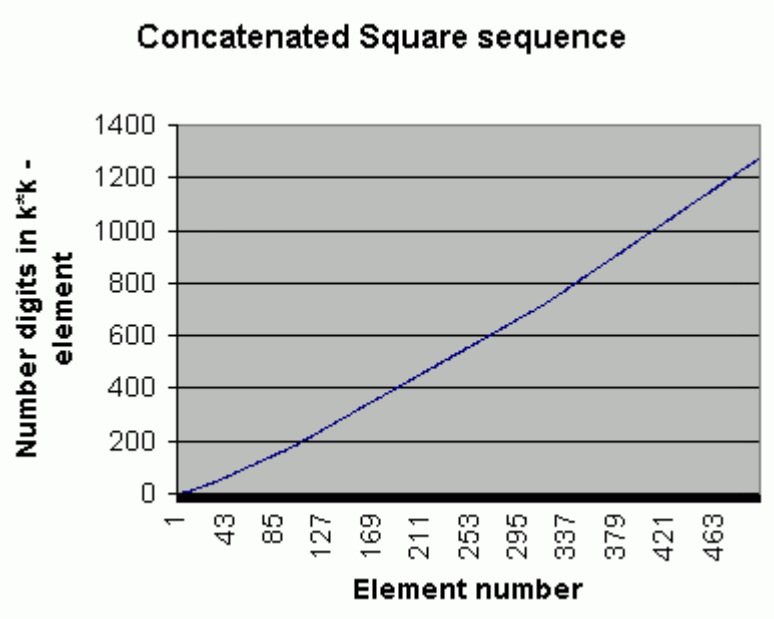


Figure 2



Question: What is the rate of growth of Diffsquare(n)?

Definition: Define the function DiffUppersquare(n) on the natural numbers to be the difference between the smallest perfect square that is greater than or equal to the nth element of the Concatenated Square sequence and the nth element of the sequence.

Question: Is DiffUppersquare(n) an always increasing function?

Question: What is the rate of growth of DiffUppersquare(n)?

While there are differences in the numbers, the general growth rate of both functions is the same.

5) Smarandache Concatenated Cubic Sequence

1, 18, 1827, 182764, 182764125, 182764125216, . . .

where element k is formed by concatenating the sequence of cubes on the right side

1(2*2*2)(3*3*3) . . . (k*k*k)

The question that was then posed was:

How many of the elements are perfect cubes?

A program was also written to search for elements of this sequence that are perfect cubes. That program is similar to the one that searches the Concatenated Square sequence for perfect squares and was used to examine the first 500 elements of the Concatenated Cube sequence. No perfect cubes were found.

As was done for the Concatenated Square problem, the differences between the element of the Concatenated Square sequence and the largest perfect square less than or equal to the element were computed. Once again, the number of digits in the difference were computed and plotted and the results are summarized in figure 3.

The program was then altered to examine the differences between the smallest perfect cube that is greater than an element of the Concatenated Cube sequence and the element of the sequence. It was also run for the first 500 elements of the sequence and the results are summarized in figure 4.

Note that in both cases, the function grows at a logarithmic rate. In fact, on closer examination it can be seen that the rate of growth for the Concatenated Cubic sequence is greater than that for the Concatenated Square sequence. Although there are some differences in the numbers, the general behavior of both graphs for the Concatenated Cube sequence is the same. Since both are increasing, the following conjecture appears to be a very safe on.

Conjecture: There is no element of the Concatenated Cube sequence which is a perfect cube.

Figure 3

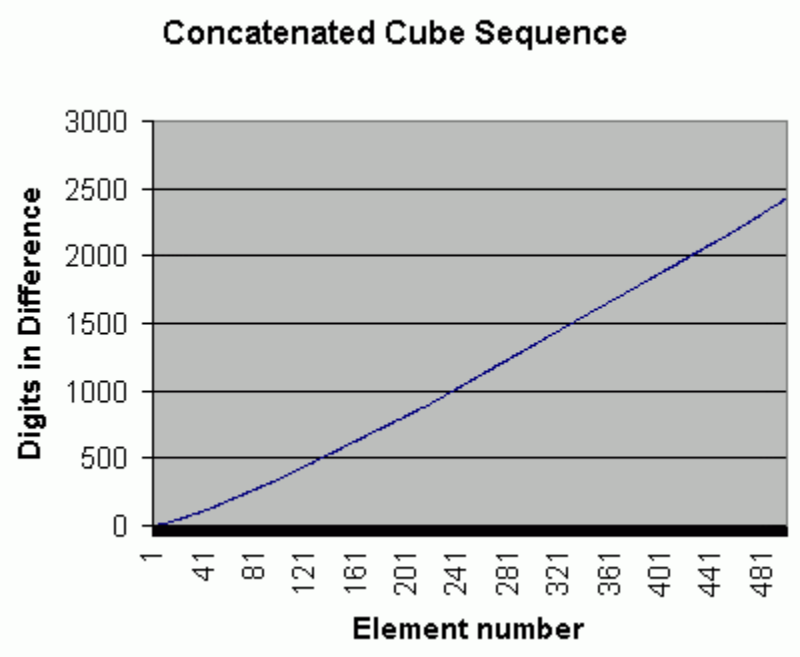
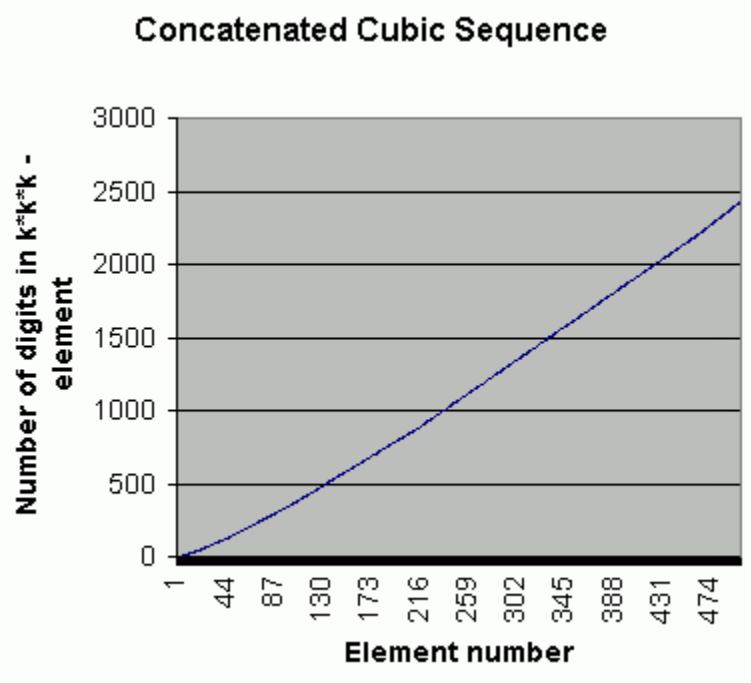


Figure 4



5) Smarandache Concatenated Fibonacci sequence

1, 11, 112, 1123, 11235, 112358, 11235813, . . .

where element k is formed by concatenating on the right the first k Fibonacci numbers.

Smarandache then asked the question:

Are any of the numbers in the Concatenated Fibonacci sequence (other than the first) a Fibonacci number?

The answer to this is almost certainly no. A program was written to examine the elements of the Fibonacci sequence looking for the first one whose prefix matches the first few digits of the Smarandache Concatenated Fibonacci sequence. For example, the first element of the Fibonacci sequence where the first two digits are 11 is element number 45, which has 10 digits. The first element of the Fibonacci sequence where the first three digits are 112 is element number 356 which has 75 digits. This was continued for the first few initial segments, and the results are summarized in table 6.

The question mark in the last row is due to the fact that no element of the Fibonacci sequence for index less than or equal to 100 has 112358 as the first 6 digits. The rate of increase in the length of the first element having the specified k digits as a prefix is summarized in figure 5, where 100,000 is taken as the value for the last entry. Note the

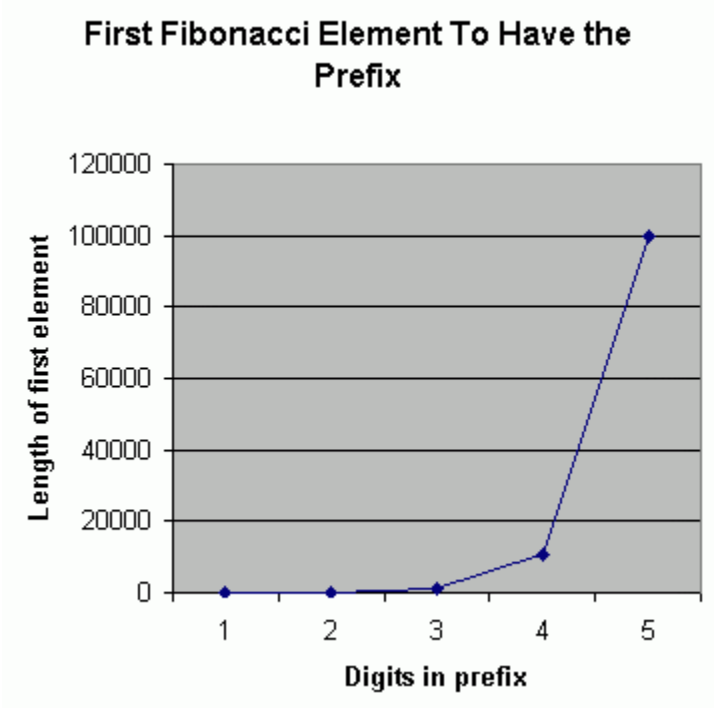
extremely rapid growth of the function. With the exception of the first two elements, there can be only one k-digit Fibonacci number that begins with a 1. Therefore, if we look at element number 51938, which has 10855 digits, for it to be a Fibonacci number, the remaining 10850 digits would all have to match those of the single Fibonacci number that begins with a 1 having that number of digits. Using the basic probability that there is a 1/10 chance of any digit position having a match, the probability that the remaining 10850 digits all match is (1/10) to the 10850 power. Therefore, the following conjecture is a very safe one.

Conjecture: Other than the first, no element of the Smarandache Concatenated Fibonacci sequence is a Fibonacci number.

Table 6

Initial digit sequence	Element index where this first appears	Number digits in element
11	45	10
112	356	75
1123	4050	847
11235	51938	10855
112358	>100,000	?

Figure 5



6) Smarandache Concatenated Lucas sequence

The Lucas sequence is one that has a definition similar to the Fibonacci sequence, the only difference is the values of the two initial elements.

$L(1) = 2, L(2) = 1, L(n) = L(n-1) + L(n-2)$ for $n > 2$

Therefore, the concatenated Lucas sequence is

2, 21, 213, 214, 2145, 21459, 2145914, ...

and we can ask the companion question:

Are any elements of the Smarandache Concatenated Lucas sequence (other than the first) Lucas numbers?

As was the case with the Fibonacci sequence, a search was performed to determine the first element of the Lucas sequence that has a specific digit prefix, and the results appear in table 7. Once again, a search of the elements through number 100,000 did not find one whose first digits are 2134711. The growth of the values is plotted in figure 6.

Table 7

Initial digit sequence	Element index where this first appears	Number digits in element
21	60	13
213	60	13
2134	950	199
21347	30880	6454
2134711	> 100,000	?

From the behavior of this function, it is clear that a similar conjecture can be made concerning the Smarandache Concatenated Lucas sequence.

Conjecture: Other than the first, no element of the Smarandache Concatenated Lucas sequence is a Lucas number.

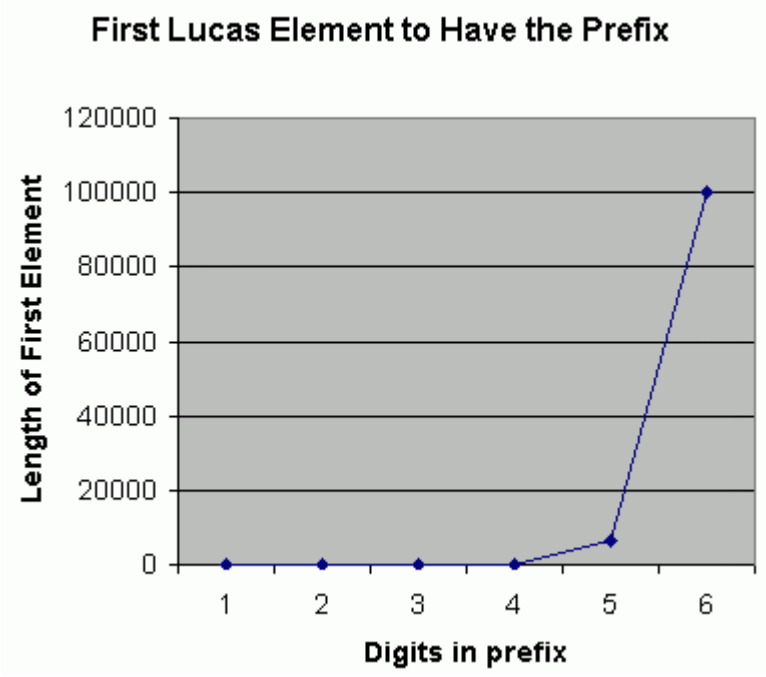
7) The generalized Fibonacci sequence

In looking at the results for the concatenated Fibonacci and Lucas sequences, the patterns for the first appearance of a specific prefix are similar. Therefore, it would appear that the behavior would be the same for all sequences with similar definitions.

The generalized Fibonacci sequence uses the same recursive summation to define all elements past the second and has arbitrary values for the first and second elements.

$G_1 = a \quad G_2 = b \quad G_n = G_{n-1} + G_{n-2}$

Figure 6



The elements of any type of Generalized Fibonacci sequence can be used to create another Concatenated sequence. For example, if a = 3 and b = 1, the sequence is

3, 1, 4, 5, 9, 14, 23, 37, . . .

and the Concatenated sequence is

3, 31, 314, 315, 3159, . . .

If we restrict the values of a and b to single digit numbers, then we can ask a question that is a companion to the one for the Fibonacci and Lucas numbers.

Question: Is there a Generalized Fibonacci sequence with $0 < a, b < 10$, such that there is a nontrivial element of the corresponding Concatenated sequence that is an element of the Generalized Fibonacci sequence?

In checking some of sequences having the properties listed in the question, it appears that the answer is no. Those sequences exhibit the same general behavior as the Fibonacci and Lucas sequences.

References

1. **Unsolved Problems in Number Theory**, edited by Richard K. Guy, Springer, 2004, ISBN 0387208607.

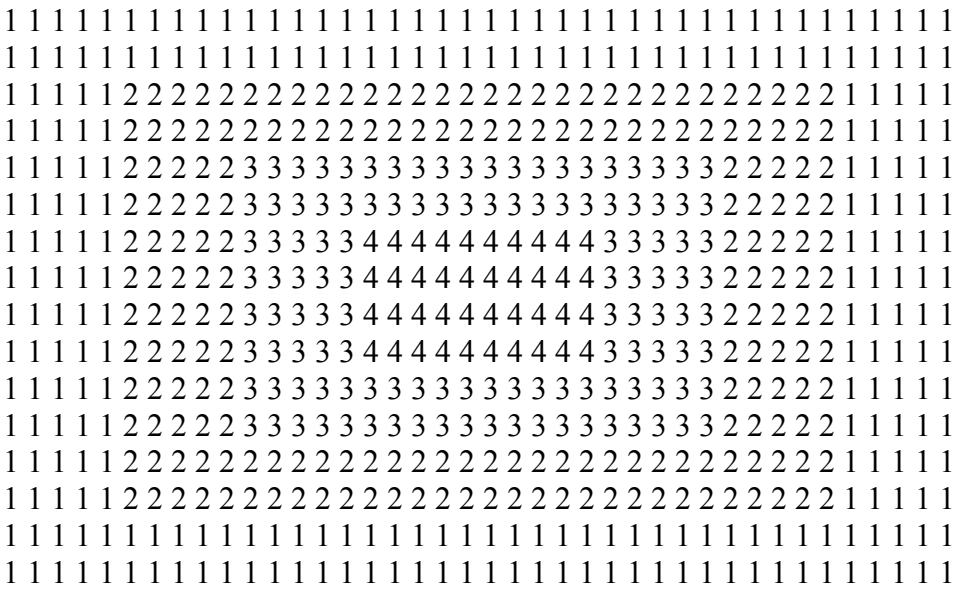
Chapter 3
Smarandache Stereograms

3.1 THE STEREOGRAM

A stereogram is a flat picture that appears three-dimensional when viewed in the correct way. For many, discerning the "hidden" image requires a great deal of patience and staring at it for some time before your eyes and brain focus on the proper characteristics of the image. A cheap, yet very good book about stereograms is "Create Stereograms on Your PC", by Paul Richardson and published by the Waite Group Press in 1994.

While most of the most stunning stereograms are developed in color, it is possible to build them using simple ASCII characters. The three-dimensional appearance of the image is created by choosing the appropriate characters and selectively altering the spaces between them. One of the simplest stereograms is a pyramid constructed using the digits 1 through 4, and is given in figure 1.

Figure 1



When you look at this figure for a length of time, the eye is drawn to the central block of fours, which looks like a cap on the top of a pyramid. This figure demonstrates one of the ways in which stereograms are presented, in that if the eye is drawn to several different locations simultaneously, it can become overloaded and "see" structure that does not really exist.

F. Smarandache defined several ASCII images that have the appearance of stereograms. By embedding the elements of a sequence inside blocks of zeros, he created two images that appear to be spiral structures that expand from top to bottom.

The first was created using the sequence

1, 111, 11011, 111, 1, 1, 111, 11011, 1100011, 11011, 111, 1, 1, 111, 11011, 1100011, 110000011, 1100011, 11011, 111, 1, . . .

and appears in figure 2.

The second was created using the sequence

1,111,11211,111,1,111,11211,1123211,11211,111,1,111,11211,1123211,112343211,...

and appears in figure 3.

Each of these images is created by embedding a series of symmetric numbers inside the matrix of infinite zeros. The Symmetric Sequence introduced in chapter 1 is used to create the image of figure 3. Another sequence of symmetric numbers that can be used to create a stereogram is the Mirror Sequence

1, 212, 32123, 4321234, 543212345, 65432123456, 7654321234567, . . .

and the image appears in figure 4. While it is possible to see the embedded image, it is not easy to see, as the contrast between the matrix zeros and some of the digits is not great. Nevertheless, the image appears to be a series of ever larger blisters arising from the background matrix.

To improve the contrast, we will alter the sequence so that the leading and trailing digits are doubled

1, 22122, 3321233, 443212344, 55432123455, 6654321234566, 776543212345677, . . .

This image appears in figure 5. The contrast between the bubbles and the background matrix is now greater and the figure starts to appear similar to that of a bulb in the Mandelbrot set, the most famous of all fractal images. An image of the Mandelbrot set appears in figure 6.

A fractal image is one that exhibits **self-similarity**, in that the general appearance of the image is the same when one zooms in. Note all of the bulbs of various sizes on the edge of the largest figure in the Mandelbrot set. If you were to zoom in on the edge, you would see even smaller bulbs, all having the same general shape.

The Mandelbrot set is defined using the iteration of a quadratic function of complex numbers.

Figure 2

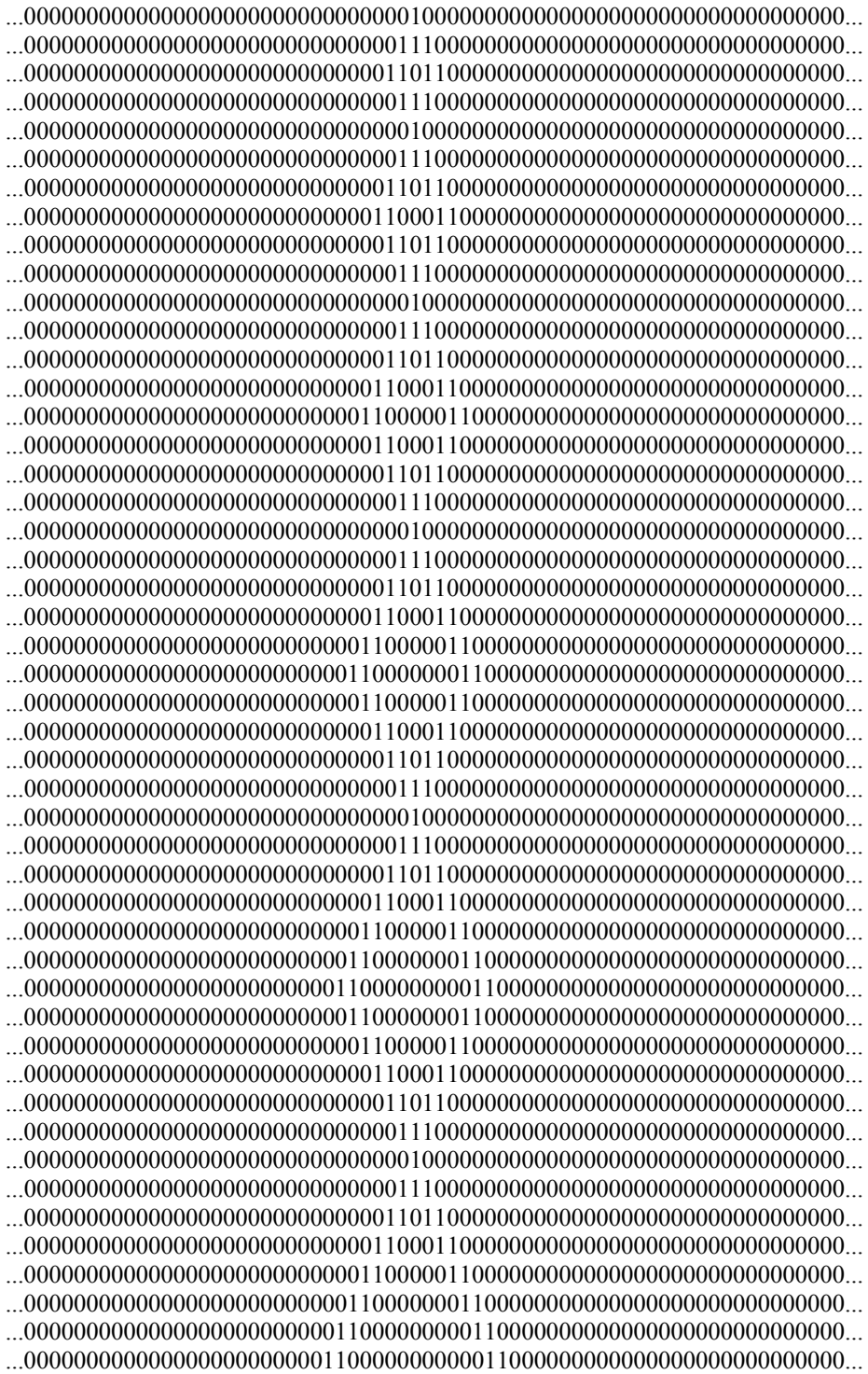

...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000000110000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000000110000000000000000000000000000...
...0000000000000000000000000000011000000011000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000001110000000000000000000000000000000000...
...0000000000000000000000000000011011000000000000000000000000000000000...
...0000000000000000000000000000011000110000000000000000000000000000000...
...0000000000000000000000000000011000011000000000000000000000000000000...
...0000000000000000000000000000011000000110000000000000000000000000000...
...0000000000000000000000000000011000000011000000000000000000000000000...
...0000000000000000000000000000011000000001100000000000000000000000000...

Figure 3

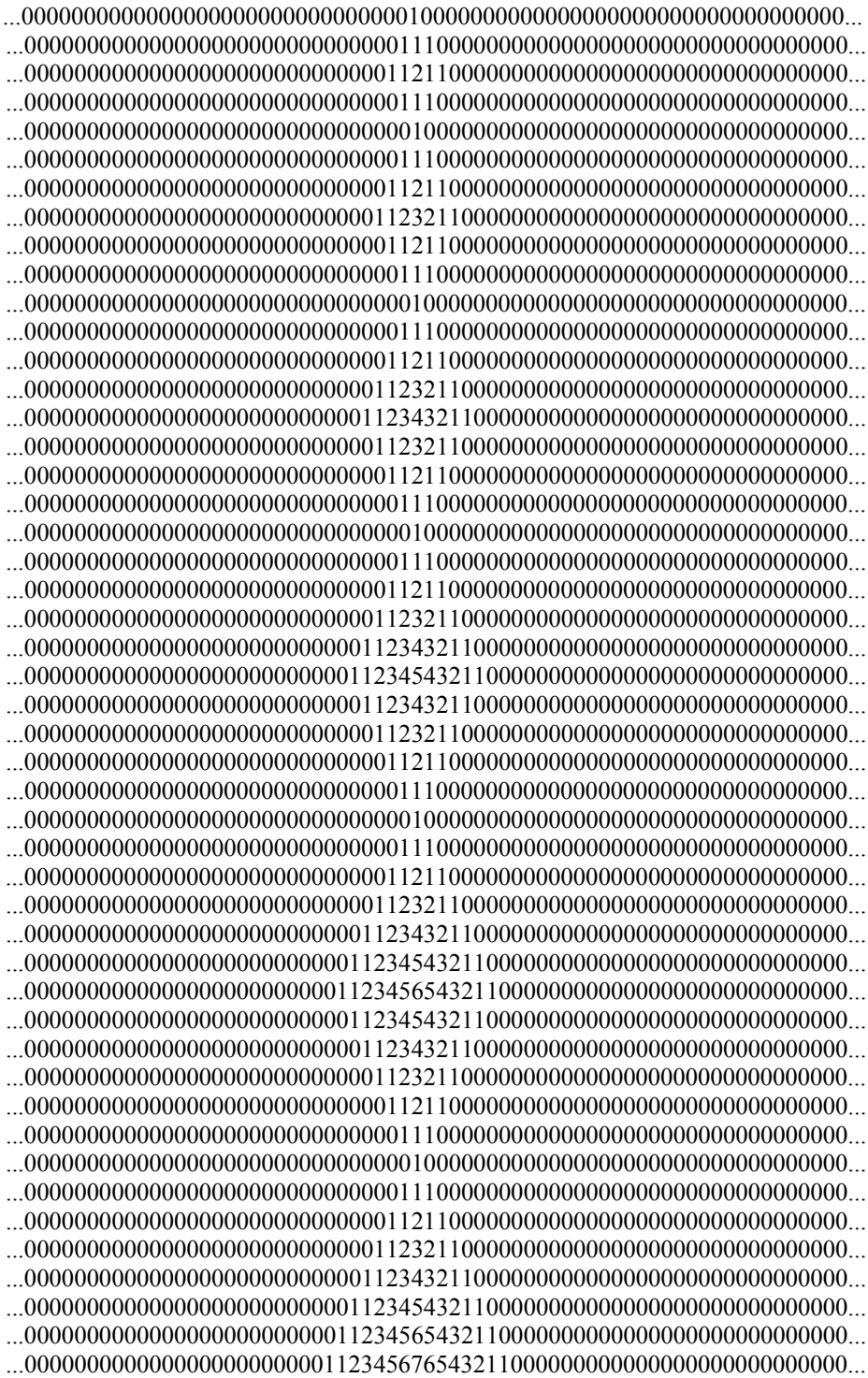


Figure 4

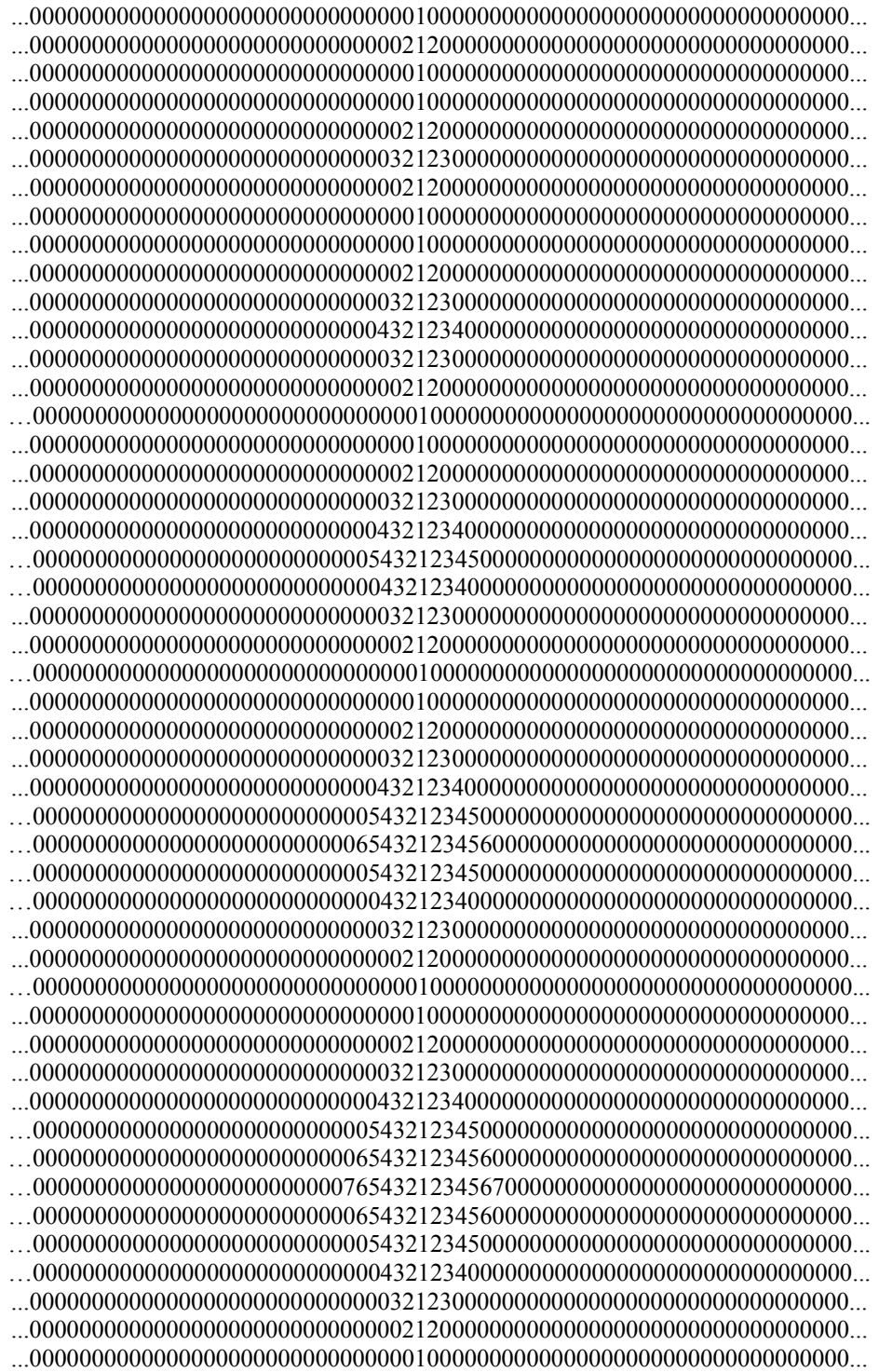

...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
…0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000000005432123450000000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
…0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000000005432123450000000000000000000000000000...
…0000000000000000000000000006543212345600000000000000000000000000...
…0000000000000000000000000005432123450000000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
…0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000000005432123450000000000000000000000000000...
…0000000000000000000000000006543212345600000000000000000000000000...
…0000000000000000000000000007654321234567000000000000000000000000...
…0000000000000000000000000006543212345600000000000000000000000000...
…0000000000000000000000000005432123450000000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000003212300000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...

Figure 5

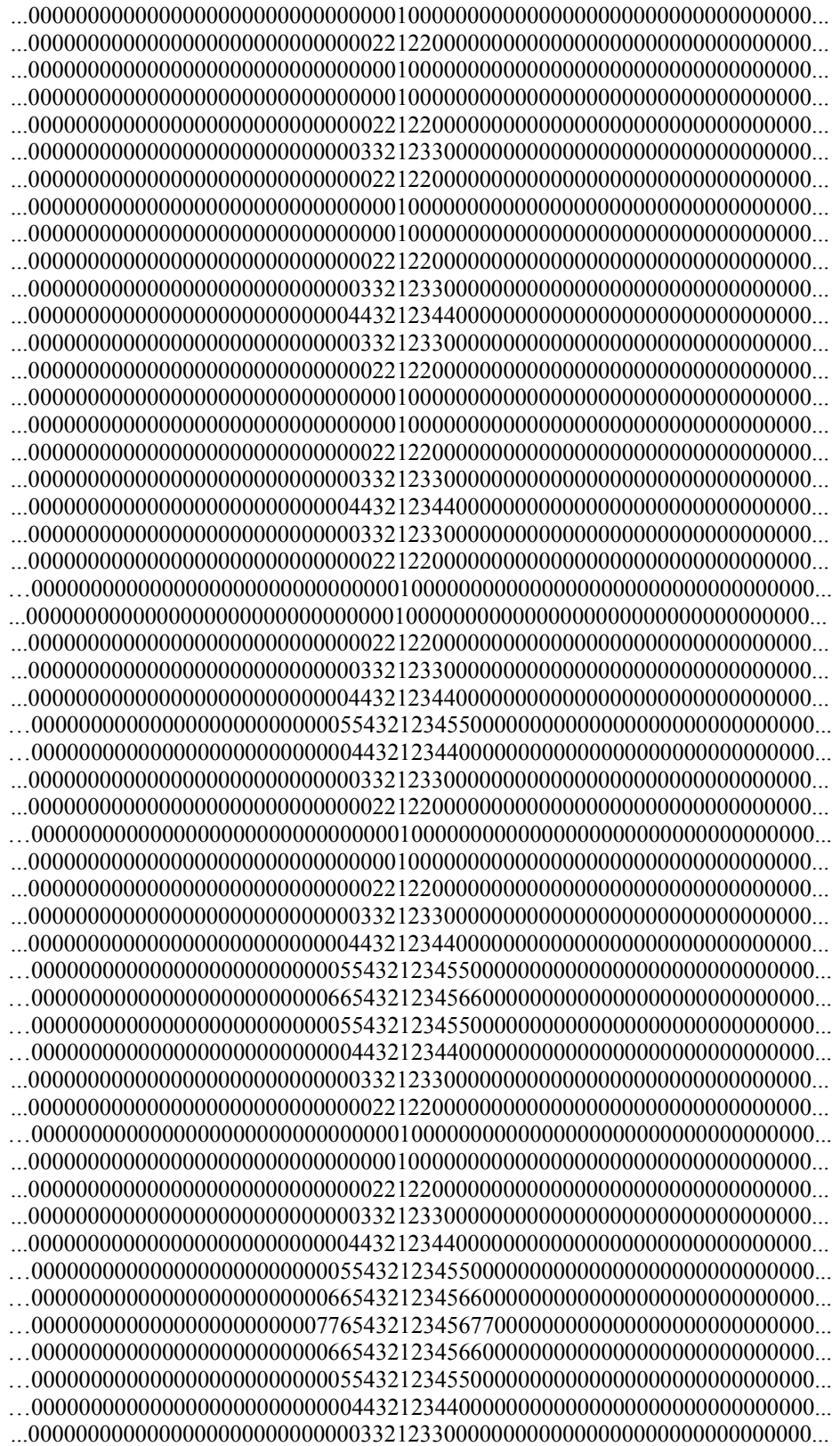
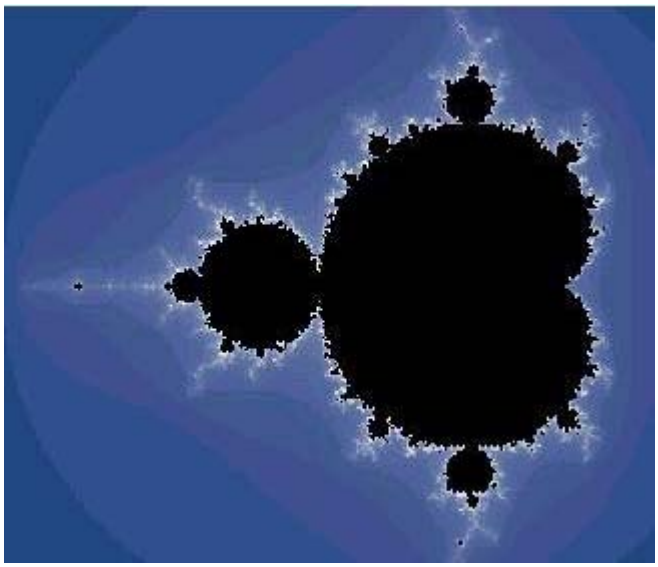

...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000004432123440000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000004432123440000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
…0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000004432123440000000000000000000000000000...
…0000000000000000000000000000005543212345500000000000000000000000000000...
…0000000000000000000000000000004432123440000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
…0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000004432123440000000000000000000000000000...
…0000000000000000000000000000005543212345500000000000000000000000000000...
…0000000000000000000000000000006654321234566000000000000000000000000000...
…0000000000000000000000000000005543212345500000000000000000000000000000...
…0000000000000000000000000000004432123440000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
…0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000000010000000000000000000000000000000000...
...0000000000000000000000000000002212200000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...
...0000000000000000000000000000004432123440000000000000000000000000000...
…0000000000000000000000000000005543212345500000000000000000000000000000...
…0000000000000000000000000000006654321234566000000000000000000000000000...
…0000000000000000000000000000007765432123456770000000000000000000000000...
…0000000000000000000000000000006654321234566000000000000000000000000000...
…0000000000000000000000000000005543212345500000000000000000000000000000...
…0000000000000000000000000000004432123440000000000000000000000000000000...
...0000000000000000000000000000003321233000000000000000000000000000000...

Figure 6



Definition: The Mandelbrot set is the set of all points c ε C where the iteration

$$z_n = z_0 + c$$

stays bounded as n $\rightarrow \infty$.

In looking at the outer bulbs of the Mandelbrot set, there are spires protruding out of the middle of the bulbs. We can mimic this in our images by creating "spikes" extending out of the middle of the bulbs. This will be done adding additional digits in the middle of the bulb, where the number of additional digits matches its value. For example, the Mirror Sequence will be modified to

1, 22122, 333212333, 4444321234444, 55555432123455555, 666666543212345666666, 777777765432123456777777, . . .

This modified sequence appears in figure 7.

The stereogram in figure 7 looks more like a bulb of the Mandelbrot set, the spikes moving out from the center are more in line with a bulb of the set.

The Concatenated natural sequence, which grows based on the repeated concatenation of a natural number with itself.

1, 22, 333, 4444, 55555, 666666, 7777777, 88888888, 999999999, 10101010101010101010, 11111111111111111111, . . .

was used to create the stereogram that appears in figure 8.

The Concatenated Square sequence, which grows by concatenating the squares of the integers on the right was used to create the image of figure 9.

The Concatenated Cubic sequence, which grows by concatenating the cubes of the integers on the right was used to create the image of figure 10.

The Concatenated Fibonacci sequence, which grows by concatenating the Fibonacci numbers on the right was used to create the image of figure 11.

Figure 7

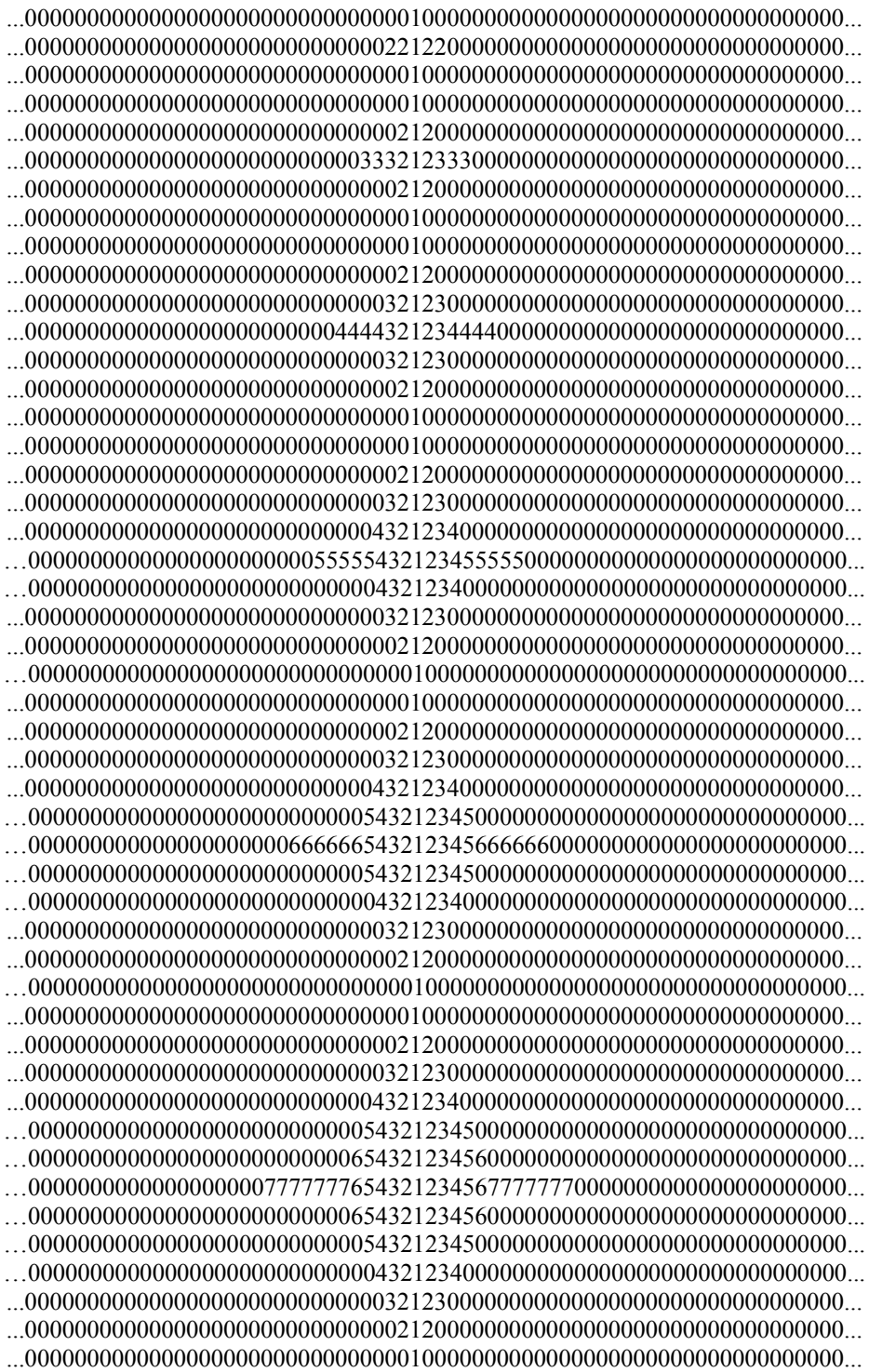

...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000022122000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000033321233000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000044443212344440000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000055554321234555550000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
…0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000000054321234500000000000000000000000000000...
…0000000000000000000006666665432123456666660000000000000000000000...
…0000000000000000000000000054321234500000000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
…0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000004321234000000000000000000000000000000...
…0000000000000000000000000054321234500000000000000000000000000000...
…0000000000000000000000000006543212345600000000000000000000000000...
…0000000000000000000077777776543212345677777770000000000000000000...
…0000000000000000000000000006543212345600000000000000000000000000...
…0000000000000000000000000054321234500000000000000000000000000000...
…0000000000000000000000000004321234000000000000000000000000000000...
...0000000000000000000000000000321230000000000000000000000000000000...
...0000000000000000000000000000021200000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000...

Figure 8

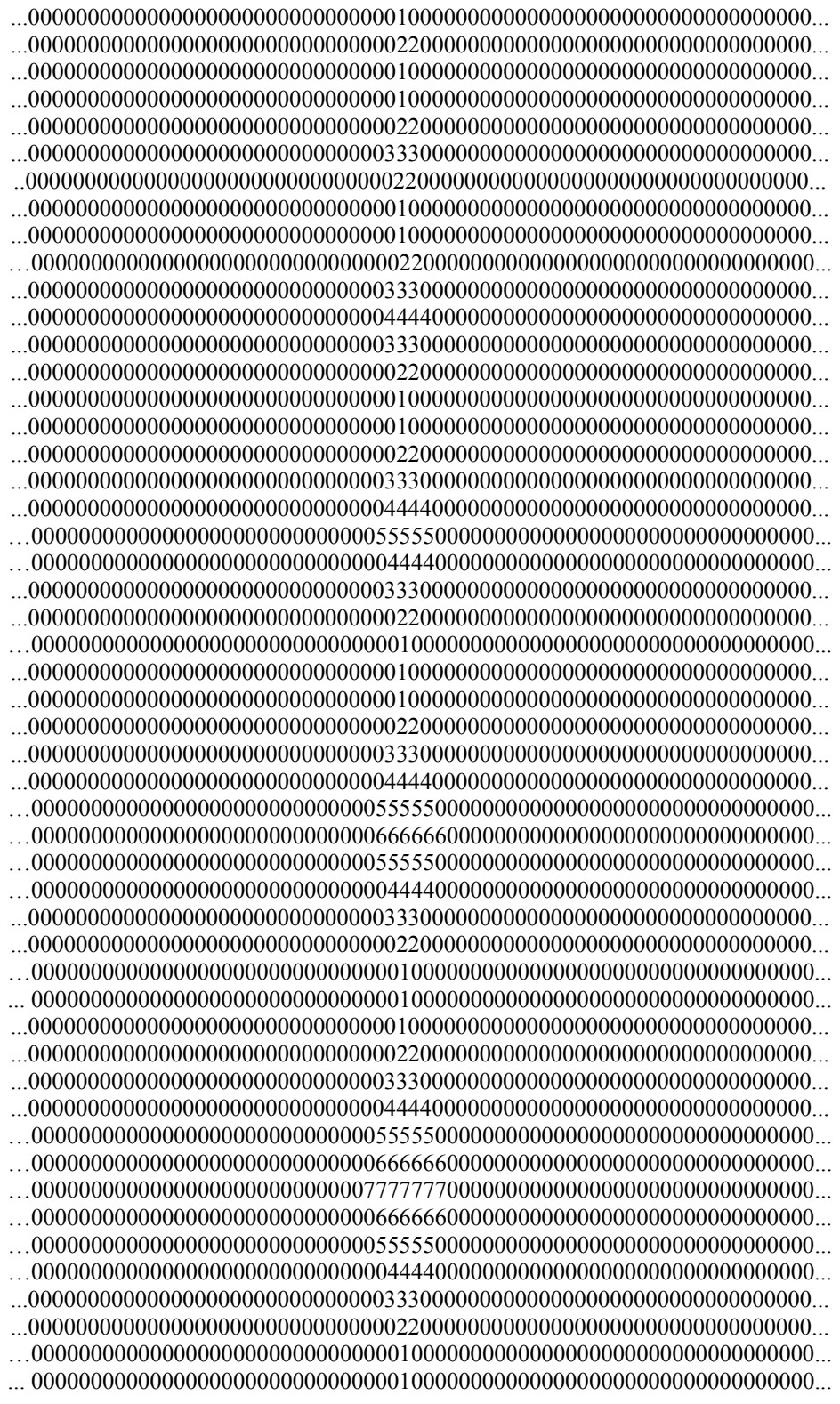

...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000220000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000220000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
..0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
…0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000004444000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000004444000000000000000000000000000000000...
…0000000000000000000000000000005555500000000000000000000000000000000...
…0000000000000000000000000000004444000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
…0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000004444000000000000000000000000000000000...
…0000000000000000000000000000005555500000000000000000000000000000000...
…0000000000000000000000000000006666660000000000000000000000000000000...
…0000000000000000000000000000005555500000000000000000000000000000000...
…0000000000000000000000000000004444000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
…0000000000000000000000000000000100000000000000000000000000000000000...
... 0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000004444000000000000000000000000000000000...
…0000000000000000000000000000005555500000000000000000000000000000000...
…0000000000000000000000000000006666660000000000000000000000000000000...
…0000000000000000000000000000007777777000000000000000000000000000000...
…0000000000000000000000000000006666660000000000000000000000000000000...
…0000000000000000000000000000005555500000000000000000000000000000000...
…0000000000000000000000000000004444000000000000000000000000000000000...
...0000000000000000000000000000003330000000000000000000000000000000000...
...0000000000000000000000000000002200000000000000000000000000000000000...
…0000000000000000000000000000000100000000000000000000000000000000000...
... 0000000000000000000000000000000100000000000000000000000000000000000...

Figure 9

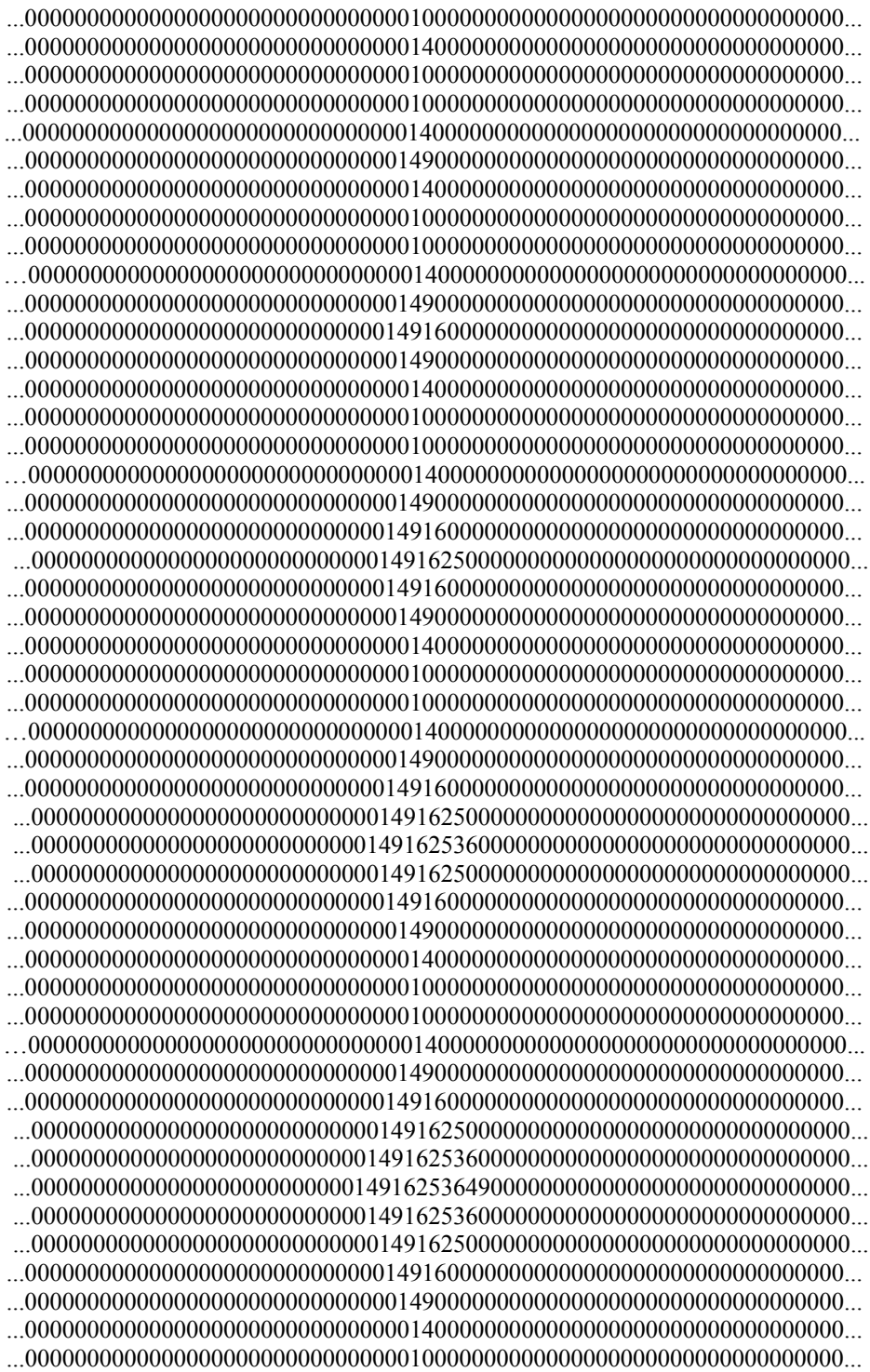

...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000140000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000140000000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
…0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
…0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001491625000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
…0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001491625000000000000000000000000000000...
...0000000000000000000000000000001491625360000000000000000000000000000...
...0000000000000000000000000000001491625000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...
…0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001491625000000000000000000000000000000...
...0000000000000000000000000000001491625360000000000000000000000000000...
...0000000000000000000000000000001491625364900000000000000000000000000...
...0000000000000000000000000000001491625360000000000000000000000000000...
...0000000000000000000000000000001491625000000000000000000000000000000...
...0000000000000000000000000000001491600000000000000000000000000000000...
...0000000000000000000000000000001490000000000000000000000000000000000...
...0000000000000000000000000000001400000000000000000000000000000000000...
...0000000000000000000000000000001000000000000000000000000000000000000...

Figure 10

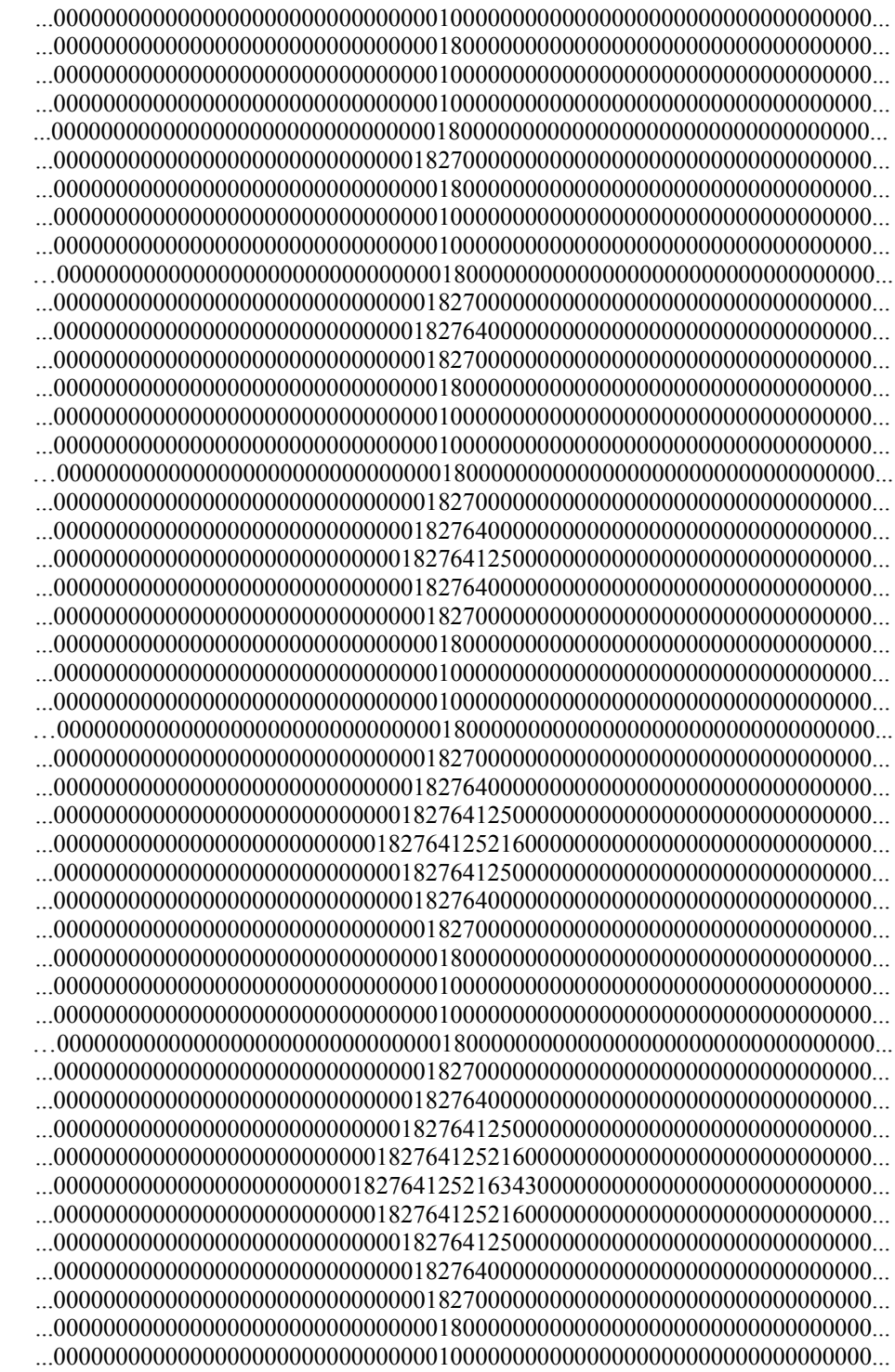
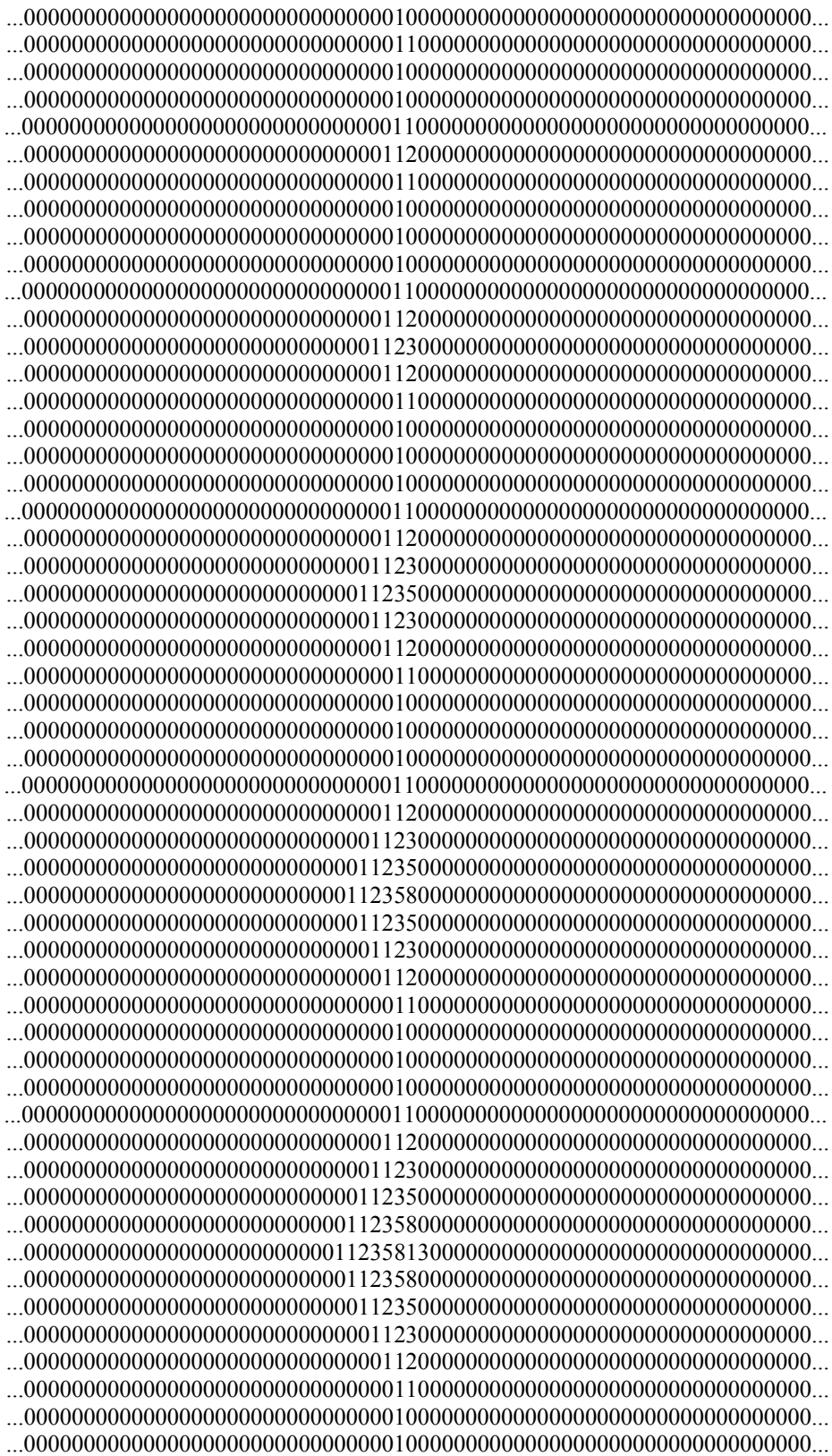

...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
…0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
…0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827641250000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
…0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827641250000000000000000000000000000...
...0000000000000000000000000000001827641252160000000000000000000000000...
...0000000000000000000000000000001827641250000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...
…0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827641250000000000000000000000000000...
...0000000000000000000000000000001827641252160000000000000000000000000...
...0000000000000000000000000000001827641252163430000000000000000000000...
...0000000000000000000000000000001827641252160000000000000000000000000...
...0000000000000000000000000000001827641250000000000000000000000000000...
...0000000000000000000000000000001827640000000000000000000000000000000...
...0000000000000000000000000000001827000000000000000000000000000000000...
...0000000000000000000000000000000180000000000000000000000000000000000...
...0000000000000000000000000000000100000000000000000000000000000000000...

Figure 11



The elements of the Smarandache Concatenated prime sequence and the Smarandache Concatenated even sequence all begin with a 2. Therefore, we can also create a combination image, where the elements of one sequence expand horizontally and the elements of the other expands vertically. The image of these two sequences in combination appears in figure 12.

The elements of the Smarandache Concatenated Odd sequence, Smarandache Concatenated square sequence, the Smarandache Concatenated Cubic sequence and the Smarandache Concatenated Fibonacci sequence all begin with a 1. Therefore, we can also create iamges where one sequence expands horizontally and the other vertically. The combination of the Smarandache Concatenated Odd sequence and Smarandance Concatenated square sequence appears in figure 13. There are few differences between figures 12 and 13, so no other sequence combinations will be done.

The Mandelbrot set is created by the repeated interation of a function and then plotting different colors based on the result. Using different colors can be considered to be a way to simulate a third dimension in the plot. We can take the grids that we have using for our plots and consider the lower left to be (0,0) and have one function define the horizontal value and another the vertical value. Each character position horizontal and vertical will be considered an integer value and in our case, we can use different digits in the positions in place of colors. Space limitations will restrict us to an 80 x 80 plot.

In his book, "Mainly Natural Numbers"[2], Henry Ibstedt discusses at length the iteration of functions on the natural numbers. Most of these functions will either enter a cycle or a fixed point when they are iterated. For example, the Pseudo-Smarandache function Z(n), which has the following definition

Z(n) is the smallest integer m such that 1 + 2 + . . . + m is divisible by n,

will enter a cycle when it is iterated. The Smarandache function S(m) has the definition

S(m) is the smallest number n such that m evenly divides n factorial.

This function has a set of fixed points consisting of the primes plus the single composite number 4. Furthermore, since $S(n) \leq n$, repeated iteration of S(n) will eventually lead to a fixed point. This is the function that we will use in our plots. Points in the quarter plane will be defined as ordered pairs of integers (m, n) and we will iterate (S(m),S(n)) until both elements of the ordered pair are fixed points. The number of iterations until a double fixed point is encountered will be counted and used to plot the point according to the following mapping.

If the number of iterations to a fixed point is 0 through 8, the character plotted in the position will be the number of iterations. A 9 will be plotted if the number of iterations is

9 or more. This iteration was done for the two dimensional grid from 1 through 50 and the results appear in figure 14.

Figure 12

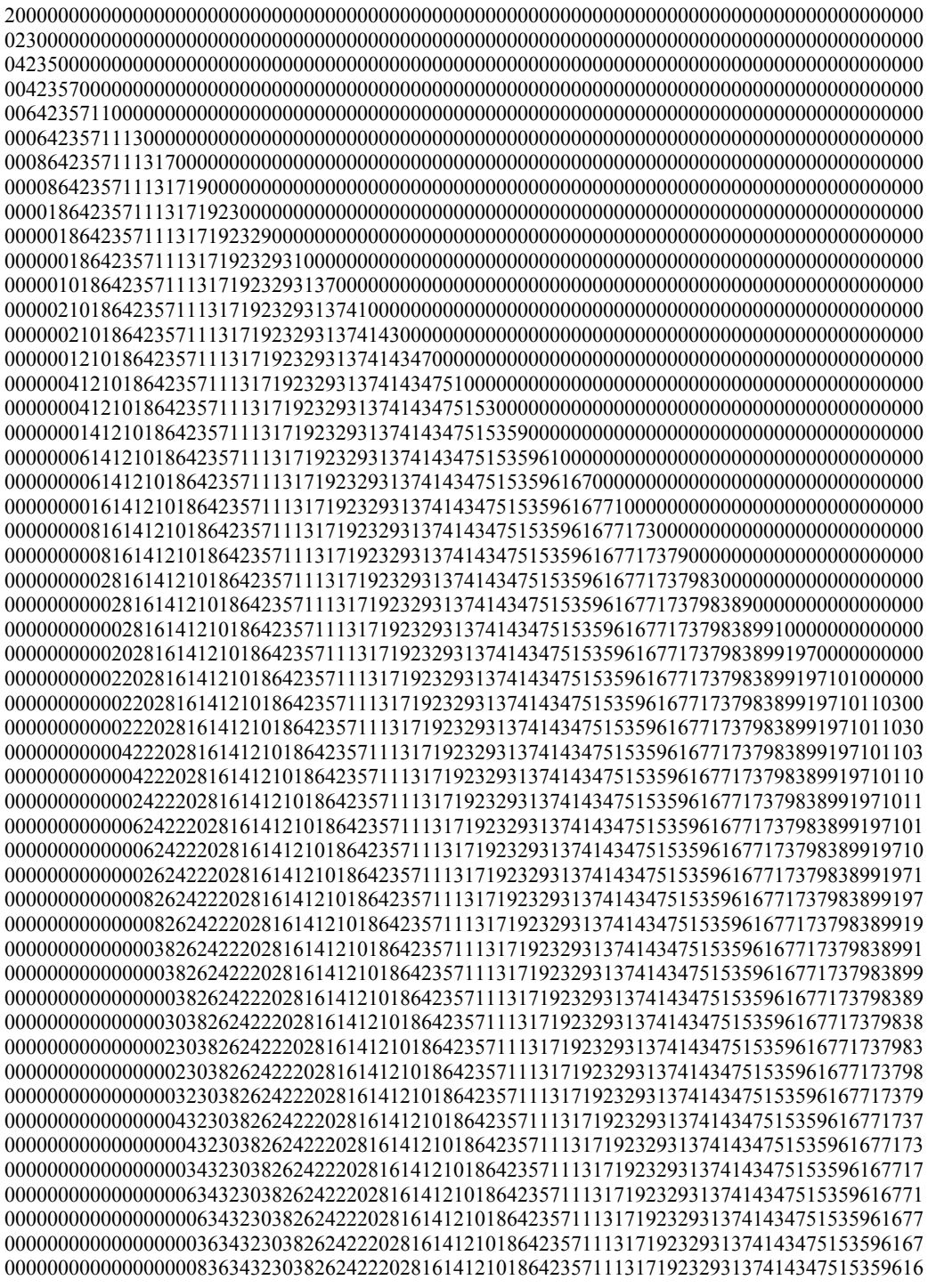


Figure 13

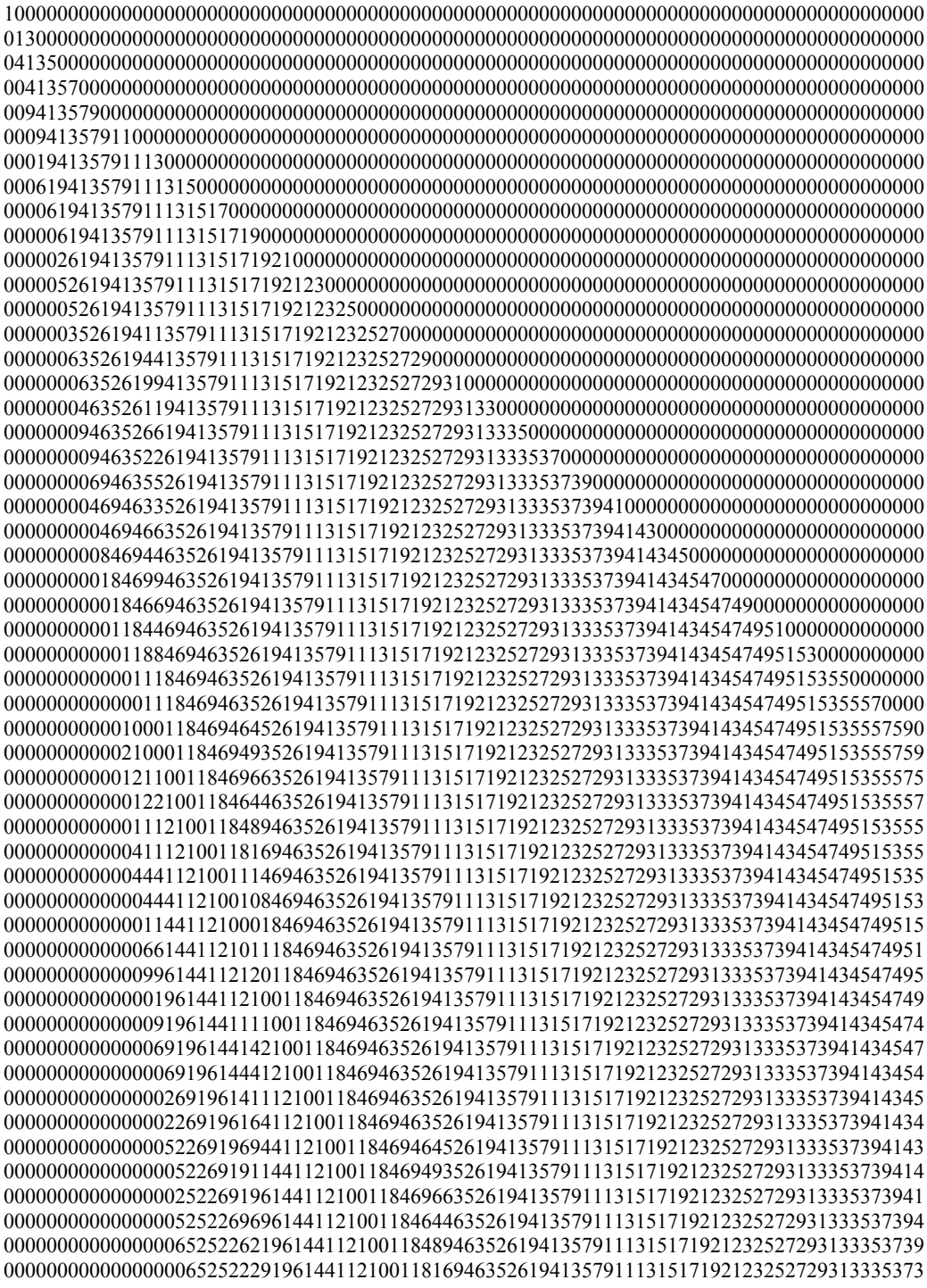
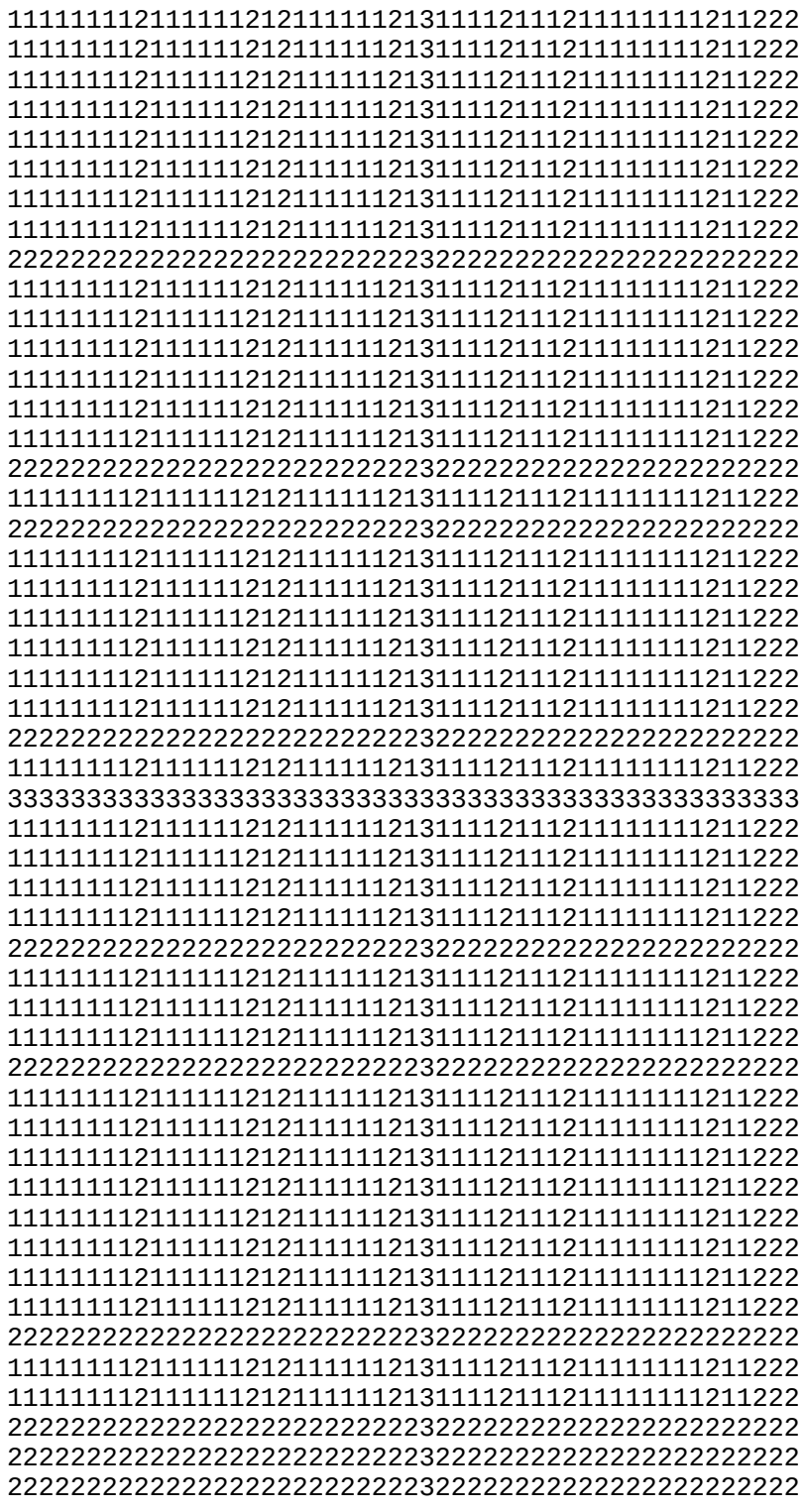

1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0130000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0413500000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0041357000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0094135790000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0009413579110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0001941357911130000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0006194135791113150000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000619413579111315170000000000000000000000000000000000000000000000000000000000000000000000000000000
0000061941357911131517190000000000000000000000000000000000000000000000000000000000000000000000000000
0000026194135791113151719210000000000000000000000000000000000000000000000000000000000000000000000000
0000052619413579111315171921230000000000000000000000000000000000000000000000000000000000000000000000
0000005261941357911131517192123250000000000000000000000000000000000000000000000000000000000000000000
0000003526194113579111315171921232527000000000000000000000000000000000000000000000000000000000000000
0000006352619441357911131517192123252729000000000000000000000000000000000000000000000000000000000000
0000000635261994135791113151719212325272931000000000000000000000000000000000000000000000000000000000
0000000463526119413579111315171921232527293133000000000000000000000000000000000000000000000000000000
0000000946352661941357911131517192123252729313335000000000000000000000000000000000000000000000000000
0000000094635226194135791113151719212325272931333537000000000000000000000000000000000000000000000000
0000000069463552619413579111315171921232527293133353739000000000000000000000000000000000000000000000
0000000046946335261941357911131517192123252729313335373941000000000000000000000000000000000000000000
0000000004694663526194135791113151719212325272931333537394143000000000000000000000000000000000000000
0000000008469446352619413579111315171921232527293133353739414345000000000000000000000000000000000000
0000000001846994635261941357911131517192123252729313335373941434547000000000000000000000000000000000
0000000000184669463526194135791113151719212325272931333537394143454749000000000000000000000000000000
0000000000118446946352619413579111315171921232527293133353739414345474951000000000000000000000000000
0000000000011884694635261941357911131517192123252729313335373941434547495153000000000000000000000000
0000000000001118469463526194135791113151719212325272931333537394143454749515355000000000000000000000
0000000000000111846946352619413579111315171921232527293133353739414345474951535570000000000000000000
0000000000010001184694645261941357911131517192123252729313335373941434547495153555759000000000000000
0000000000021000118469493526194135791113151719212325272931333537394143454749515355575900000000000000
0000000000012110011846966352619413579111315171921232527293133353739414345474951535557500000000000000
0000000000001221001184644635261941357911131517192123252729313335373941434547495153555700000000000000
0000000000001112100118489463526194135791113151719212325272931333537394143454749515355500000000000000
0000000000004111210011816946352619413579111315171921232527293133353739414345474951535500000000000000
0000000000004441121001114694635261941357911131517192123252729313335373941434547495153500000000000000
0000000000004441121001084694635261941357911131517192123252729313335373941434547495153000000000000000
0000000000001144112100018469463526194135791113151719212325272931333537394143454749515000000000000000
0000000000000661441121011184694635261941357911131517192123252729313335373941434547495100000000000000
0000000000000996144112120118469463526194135791113151719212325272931333537394143454749500000000000000
0000000000000019614411210011846946352619413579111315171921232527293133353739414345474900000000000000
0000000000000091961441111001184694635261941357911131517192123252729313335373941434547400000000000000
0000000000000069196144142100118469463526194135791113151719212325272931333537394143454700000000000000
0000000000000006919614441210011846946352619413579111315171921232527293133353739414345400000000000000
0000000000000002691961411121001184694635261941357911131517192123252729313335373941434500000000000000
0000000000000002269196164112100118469463526194135791113151719212325272931333537394143400000000000000
0000000000000005226919694411210011846946452619413579111315171921232527293133353739414300000000000000
0000000000000000522691911441121001184694935261941357911131517192123252729313335373941400000000000000
0000000000000000252269196144112100118469663526194135791113151719212325272931333537394100000000000000
0000000000000000525226969614411210011846446352619413579111315171921232527293133353739400000000000000
0000000000000000652522621961441121001184894635261941357911131517192123252729313335373900000000000000
0000000000000000065252229196144112100118169463526194135791113151719212325272931333537300000000000000

Figure 14


11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
33333333333333333333333333333333333333333333333333
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
11111111211111121211111121311112111211111111211222
11111111211111121211111121311112111211111111211222
22222222222222222222222223222222222222222222222222
22222222222222222222222223222222222222222222222222
22222222222222222222222223222222222222222222222222

The patterns that appear when the Smarandache function is iterated is expected, in that the number of iterations for the pair will be the largest number of iterations that it will take for the individual elements of the pair. However, it does give us a place to start, in that there are many different functions that can be applied.

For example, the iteration cycle could be altered by starting with (Z(m), Z(n)) instead of (m,n) and iterating the Smarandache function as before. The results of this action are demonstrated in figure 15.

There are many different combinations of functional iterations that can be done, for example both Ashbacher[3] and Ibstedt[2] have iterated combinations of the Smarandache, Pseudo-Smarandache, sum-of-divisors, Euler phi and number of factors functions. Therefore, we conclude this segment with some questions.

Question: There are many different functions that define sequences that can be placed inside a matrix of zeros. Which of these functions can be used to create the stereogram that most accurately resembles the Mandelbrot set?

Question: What kind of stereograms can be created by iterating combinations of number theory functions and plotting the number of iterations?

3.2 Altering the Background

Smarandache defined additional images using some of the same infinite sequences, only now the matrix is created using ones rather than zeros. This alters the fundamental appearance of the matrix, but with the exception of the leading and trailing ones, the "bulbs" remain the same. Two of these stereograms appear in figures 16 and 17.

In looking at these stereograms, while there is some difference in the appearance, the general structure is still the same. However, we can use it to ask a more general question regarding the Mandelbrot set.

Question: There are many different functions that define sequences that can be placed inside a matrix composed of a single digit chosen from { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }. Which of these functions and choice of digit can be used to create the stereogram that most accurately resembles the Mandelbrot set?

Figure 15

```
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
222222222222222222222222222222222222222222222222222223222222
222222222222222222222222222222222222222222222222222223222222
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
111111111111111121211111111111111211211111122112211113122111
333333333333333333333333333333333333333333333333333333333333
```

Figure 16

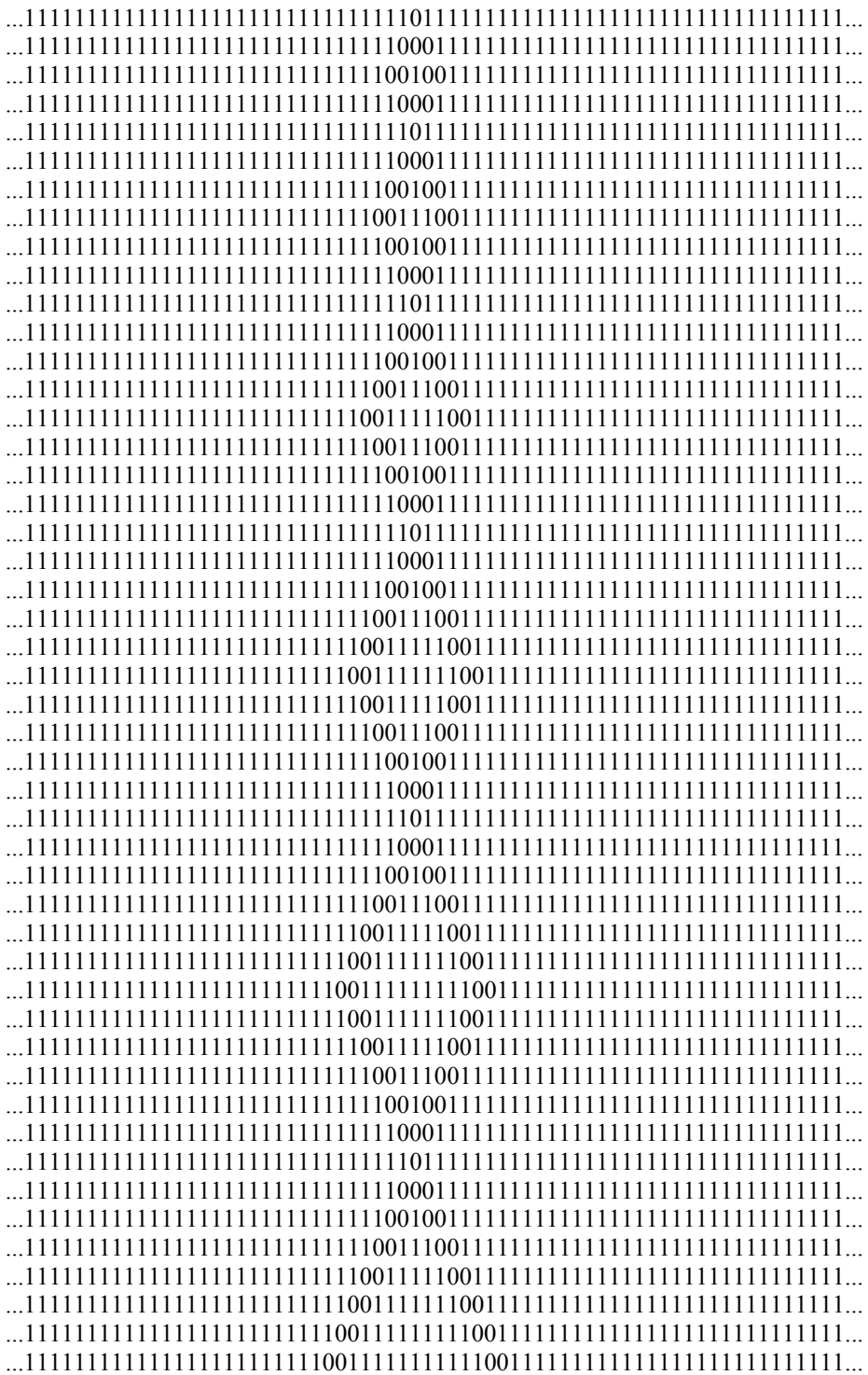


Figure 17

...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111112234565432211111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111112234565432211111111111111111111111111111...
...11111111111111111111111122345676543221111111111111111111111111111...
...11111111111111111111111112234565432211111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111111121111111111111111111111111111111111...
...11111111111111111111111111111222111111111111111111111111111111111...
...11111111111111111111111111112232211111111111111111111111111111111...
...11111111111111111111111111122343221111111111111111111111111111111...
...11111111111111111111111111223454322111111111111111111111111111111...
...11111111111111111111111112234565432211111111111111111111111111111...
...11111111111111111111111122345676543221111111111111111111111111111...
...11111111111111111111111223456787654322111111111111111111111111111...

3.3 Additional Stereograms

Smarandache also used the decimal digits to create other patterns inside a matrix constructed from a single digit. In these cases, the patterns are made without the use of a sequence. Three of these constructions appear in figures 18, 19 and 20.

Figure 18
Car

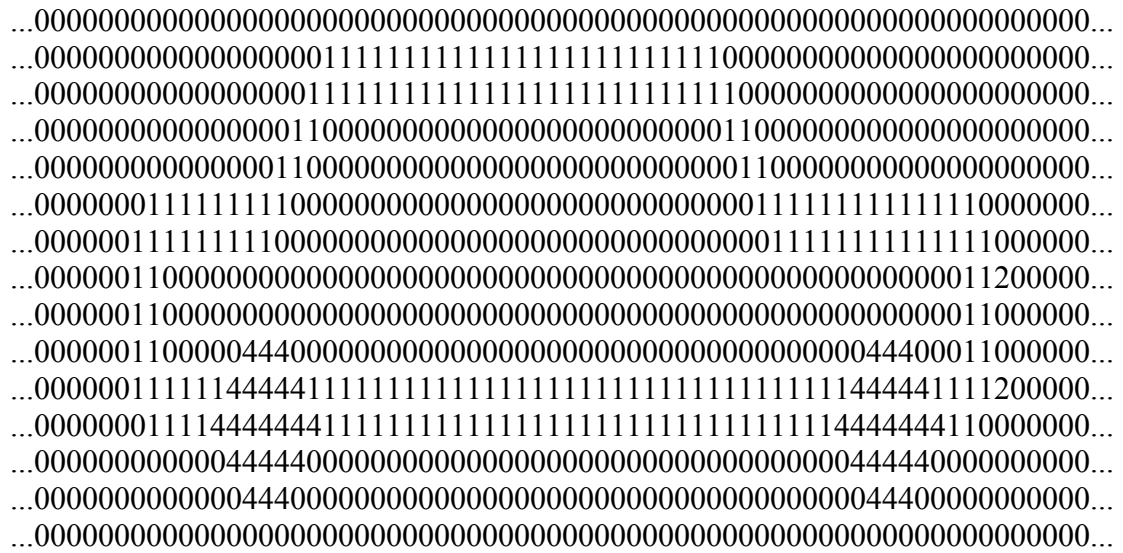


Figure 19
Finite Lattice

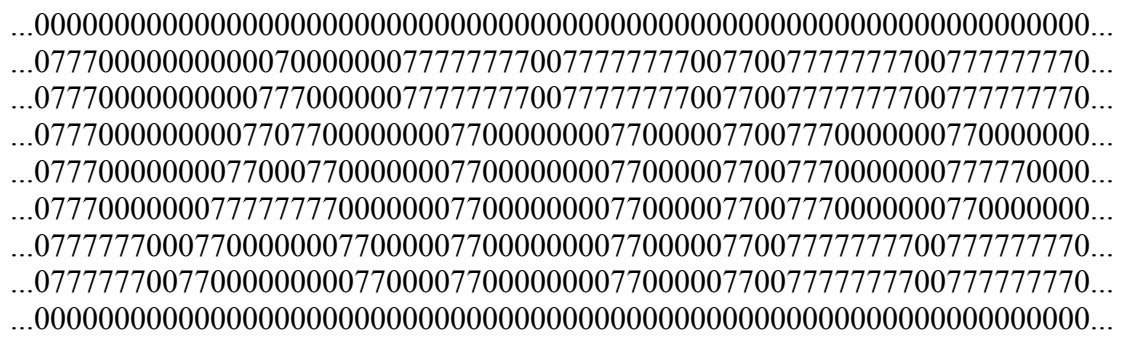
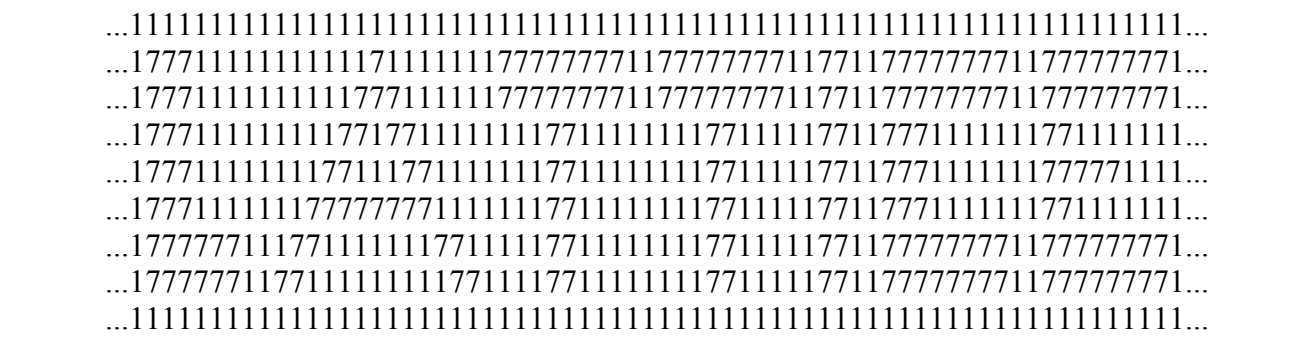


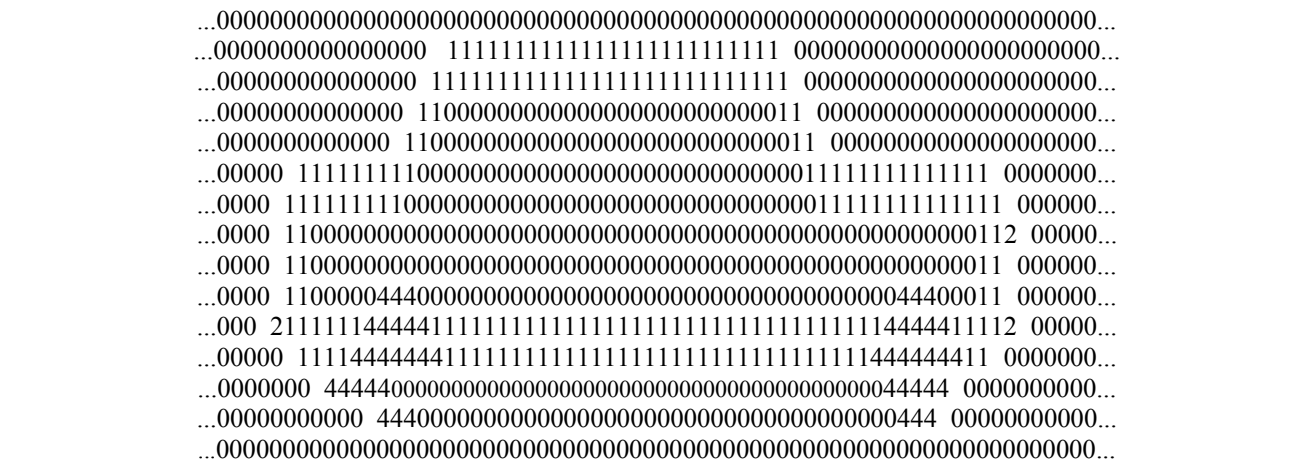
Figure 20
Infinite Lattice



The difference between figure 2 and figure 16 is that the background matrix of zeros has been replaced by a background matrix of ones. This distinction was made by Smarandache.

By adding spaces around the car, it is possible to make it more distinctive, rendering the appearance more like that of an automobile. In figure 21, the size of the zeros above and below the car have been reduced to create some additional whitespace separation.

Figure 21



3.4 Possibilities For Additional Figures

Using mathematical expressions and formulas to create text-based stereograms is clearly an area where many things can be done. There are many alterations that could be performed on the figures in this chapter. The inclusion of additional whitespace and using different digits for the figure and the matrix could lead to either small or significant changes in the appearance.

Figure 22

...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 22122 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 333212333 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 4444321234444 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 4321234 0000000000000000000000000000000...
…0000000000000000000000 5555543212345555 0000000000000000000000000...
…0000000000000000000000000000000 4321234 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
…0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 4321234 0000000000000000000000000000000...
…0000000000000000000000000 543212345 0000000000000000000000000000000...
…0000000000000000000 666666543212345666666 0000000000000000000000...
…0000000000000000000000000000000 543212345 0000000000000000000000000000000...
…0000000000000000000000000000000 4321234 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
…0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 4321234 0000000000000000000000000000000...
…0000000000000000000000000 543212345 0000000000000000000000000000000...
…0000000000000000000000000 65432123456 0000000000000000000000000000000...
…0000000000000000000 77777776543212345677777777 0000000000000000000000...
…0000000000000000000000000 65432123456 0000000000000000000000000000000...
…0000000000000000000000000 543212345 0000000000000000000000000000000...
…0000000000000000000000000000000 4321234 0000000000000000000000000000000...
...0000000000000000000000000000000 32123 0000000000000000000000000000000...
...0000000000000000000000000000000 212 0000000000000000000000000000000...
...0000000000000000000000000000000 1 0000000000000000000000000000000...

For example, figure 22 is simply figure 7 with whitespace placed around the figure. This is another one of those areas where F. Smarandache offered a starting point, and there are many different directions that can be followed.

References

1. D. Richardson, “Create Stereograms on Your PC: Discover the World of 3D Illusion”, The Waite Group Press, Corte Modera, CA, 1994.
2. H. Ibstedt, “Mainly Natural Numbers”, Bookman Publishing, Martinsville, IN, 2003.
3. C. Ashbacher, “Pluckings From the Tree of Smarandache Sequences and Functions”, American Research Press, 1998.

Chapter 4

Constants Involving the Smarandache Function

F. Smarandache defined a collection of infinite series where the terms are computed using the Smarandache function.

Definition: Let n be a positive integer, then S(n) is the smallest integer m such that n evenly divides m!.

In this chapter, we will present some basic computations concerning the values of the infinite series. When the series is introduced, the style will be that used when the series were defined.

1) The first constant of Smarandache

$$\sum_{n=2}^{\infty} \frac{1}{S(n)!}$$

is convergent to a number s1 between 0.000 and 0.717.

The fact that the sum is convergent is easy to prove and is based on the following theorem.

Theorem: For $n > 10$, $S(n)! > n$.

Proof:

Case 1: n is the product of k distinct primes, i. e. $n = p_1 p_2 \ldots p_k$, where the primes are in ascending order. Then $S(n) = p_k$ and S(n)! contains all of the other primes and composite numbers as factors. Therefore, $S(n)! > n$.

Case 2: Express n in the prime factorization form, $n = p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_k^{\alpha k}$. By definition $S(n) = rp_j$, where p_j is the prime of concern. Again, by definition rp_j is the smallest number such that the factorial contains all the proper numbers of primes in the prime factorization. Since the factorial must contain a prime not in the prime factorization, it follows that $S(n)! > n$.

To examine this and all the other infinite series in more detail, a computer program was written in the Java language. The language contains the BigDecimal predefined data type, which allows for a nearly unlimited number of decimal points of storage. Therefore, the only limits to the number of decimal places in our computation are the memory on the platform and our patience.

The program used to compute the values of the first constant appears in listing 1. Since the long integer in Java has the upper limit 9223372036854775807, our range of computation is well within this bound. However, we use an unlimited precision BigInteger for the computation of the factorial.

Listing 1

```
/* This program will compute the sum of the series Sigma(1/S(n)!) for n through
   THELIMIT. The result will then be placed in the file ConstantSums.txt. Written by
   Charles Ashbacher. */

import java.io.*;
import java.math.*;

public class SumConstant1
{
// The upper limit to the sums.
 static final long THELIMIT=10000;

// This is the number of digits of precision to use in the BigDecimal computations.
 static final int THEPRECISION=2000;

// Space for the factor and exponent pairs when the number is factored.
 static long []theExponents=new long[1000];
 static long []theFactors=new long[1000];

// This will be the values of the Smarandache function for each factor exponent pair.
 static long []smars=new long[1000];

// The number of factors in the number.
 static int FactorsCount;

// References for the output file.
 static String filename="ConstantSums.txt";
 static File output;
 static BufferedWriter out;

// This function converts the input long data value into the prime factors. The factor
```

```
// will be placed in the theFactors[] array and the corresponding exponent in the
// theExponents array. The special cases of zero and one are dealt with separately.

public static void GetFactors(long theValue)
{
 int i1,j1;
 long temp1;

// Initialize the exponents and the count of factors.
 FactorsCount=0;
 for(i1=0;i1<1000;i1++)
 {
  theFactors[i1]=0;
  theExponents[i1]=0;
 }

// An integer must be larger than one before it can have a true prime factorization.
 if(theValue>1)
 {

// Factor out all twos first
  temp1=theValue;
  if((temp1%2)==0)
  {
   FactorsCount=1;
   theFactors[0]=2;
   while((temp1%2)==0)
   {
    theExponents[0]=theExponents[0]+1;
    temp1=temp1/2;
   }
  }

// Factor out all odd numbers. Note that no attempt is made to filter out the non-primes.
// In this case, it makes sense, because no nonprime at this point could divide the
// number, so the if will filter them out. This is much faster than testing the primeness
// of the possible factor.

  j1=3;
  while(temp1>1)
  {
   if((temp1%j1)==0)
   {
    theFactors[FactorsCount]=j1;
    theExponents[FactorsCount]+=1;
    temp1=temp1/j1;
```

```
    while((temp1%j1)==0)
    {
     theExponents[FactorsCount]+=1;
     temp1=temp1/j1;
    }
    FactorsCount++;
   }
   j1+=2;
  }
 }
}

// This function will accept a long integer and return the factorial of that input as
// a BigInteger.

public static BigInteger Factorial(long in)
{
// Initialize the output to 1
 BigInteger theResult=new BigInteger("1");

// The input is converted into the BigInteger equivalent.
 Long theInput=new Long(in);
 BigInteger theLimit=new BigInteger(theInput.toString());

// The BigInteger theIncrement is the variable that will by multiplied with the
// accumulator.
// The process will proceed as long as theIncrement is less than or equal to the input.
 BigInteger theIncrement=new BigInteger("1");
 while(theIncrement.compareTo(theLimit)<=0)
 {
  theResult=theResult.multiply(theIncrement);
  theIncrement=theIncrement.add(BigInteger.ONE);
 }
 return theResult;
}

// This function will accept the input long integer and compute the value of the
// Smarandache function. Note that it assumes that the number has been factored and the
// prime factors are in  the two arrays theFactors and theExponents. This is done by
// calling GetFactors().

public static long SmarFunction(long theValue)
{
 int i1,found;
 long startnum,tempsum,sum_of_factor,result1;
 if(theValue>1)
```

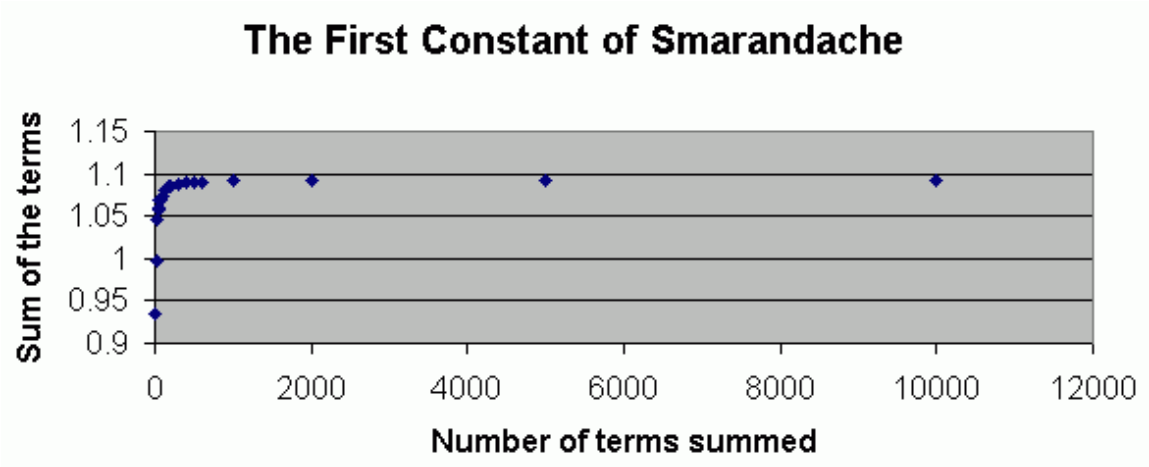
```
{
 for(i1=0;i1<1000;i1++)
 {
  smars[i1]=0;
 }
 for(i1=0;i1<FactorsCount;i1++)
 {
  if(theExponents[i1]<theFactors[i1])
  {
   smars[i1]=theExponents[i1]*theFactors[i1];
  }
  else
  {
   startnum=theExponents[i1]/theFactors[i1];
   if(startnum<1)
   {
    startnum=1;
   }
   found=0;
   while(found==0)
   {
    sum_of_factor=startnum;
    tempsum=startnum/theFactors[i1];
    while(tempsum>0)
    {
     sum_of_factor=sum_of_factor+tempsum;
     tempsum=tempsum/theFactors[i1];
    }
    if(sum_of_factor>=theExponents[i1])
    {
     found=1;
    }
    else
    {
     startnum++;
    }
   }
   smars[i1]=startnum*theFactors[i1];
  }
 }
 result1=0;
 for(i1=0;i1<FactorsCount;i1++)
 {
  if(smars[i1]>result1)
  {
   result1=smars[i1];
```

```
   }
  }
 }
 else
 {
  result1=0;
 }
 return result1;
}

// This function can be used to show the factorization of the number if needed.

 public static void ShowFactors()
 {
  for(int k1=0;k1<FactorsCount;k1++)
  {
   System.out.print(" "+theFactors[k1]);
  }
  System.out.println(" ");
  for(int k1=0;k1<FactorsCount;k1++)
  {
   System.out.print(" "+theExponents[k1]);
  }
  System.out.println(" ");
 }

 public static void main(String[] args)
 {
  long theNumber,theSmar;
  BigInteger theFactorial;

// This will be the accumulated sum of the series.
  BigDecimal theSum=new BigDecimal("0.0");

// The numerator and denominator of the fraction to be added.
  BigDecimal theDenominator;
  BigDecimal theNumerator=new BigDecimal("1.0");

// The fraction to be added to theSum.
  BigDecimal theFraction;

// Prepare the file for output.
  try
  {
   output=new File(filename);
   output.createNewFile();
```

```
  if(!output.isFile())
  {
   System.out.println("Error creating the file");
   IOException ioe=new IOException();
   throw ioe;
  }
  out=new BufferedWriter(new FileWriter(output));
 }
 catch(IOException ioe)
 {
  System.out.println("Cannot open the designated file");
  System.exit(0);
 }

 System.out.println("Summing through "+THELIMIT);
 for(long i=2;i<=THELIMIT;i++)
 {
  GetFactors(i);
  theSmar=SmarFunction(i);
  theFactorial=Factorial(theSmar);
  theDenominator=new BigDecimal(theFactorial);

// This line creates the fraction. The input THEPRECISION is how many digits of
//  precision to use in the computation.

  theFraction=theNumerator.divide(theDenominator,THEPRECISION,
                                        BigDecimal.ROUND_HALF_EVEN);

  theSum=theSum.add(theFraction);
 }
 System.out.println("Summed through "+THELIMIT);
 System.out.println("The sum is "+theSum);
  try
  {
   out.write("Summed through "+THELIMIT+"\n");
   out.write("The sum is "+theSum);
   out.close();
  }
  catch(IOException io)
  {
   System.out.println("Cannot write to the log file");
  }
 }
}
```

The program was run for several different upper limits and the plot of those results appears in figure 1.

Figure 1



The last five values plotted are summarized in table 1. From this plot, the indications are clear that the sum of the series is very close to 1.1. One final sum was done where the limit was 20,000 and the sum was 1.0930689559.

Table 1

Upper limit of sum	Sum
600	1.089654
1000	1.091555
2000	1.092296
5000	1.092700
10000	1.092996

2) The second constant of Smarandache

$$\sum_{n=2}^{\infty} \frac{S(n)}{n!}$$

is convergent to an irrational number s2.

The first step is to prove the convergence.

Theorem: The sum that defines the second constant of Smarandache is convergent.

Proof: Since S(n) = n for 4 and all primes, and is less than n for all other numbers, each term in the sequence is less than or equal to

$$\frac{n}{(n-1)!}.$$

Since the series

$$\sum_{n=2}^{\infty} \frac{n}{(n-1)!}$$

converges, we can apply the comparison test to conclude that

$$\sum_{n=2}^{\infty} \frac{S(n)}{n!}$$

also converges.

The program was modified to compute partial sums for this series and the changes are given in listing 2.

Listing 2

```
for(long i=2;i<=THELIMIT;i++)
{
 GetFactors(i);
 theSmar=new Long(SmarFunction(i));
 theNumerator=new BigDecimal(theSmar.toString());
 theFactorial=Factorial(i);
 theDenominator=new BigDecimal(theFactorial);
```

```
  theFraction=theNumerator.divide(theDenominator,
                  THEPRECISION,BigDecimal.ROUND_HALF_EVEN);
  theSum=theSum.add(theFraction);
 }
```

This series converges much more rapidly than the first constant, so the results will be listed for the first six summations, where each is an increase by ten.

Summed through 10
The sum is
1.7140060074955908289241622574955908289241622574955908289241622574955 90

Summed through 20
The sum is
1.7140062935916160225800499835714961401363317555395426179867311745664 12

Summed through 30
The sum is
1.7140062935916160227277438454190299531157681845656409273738576638482 26

Summed through 40
The sum is
1.7140062935916160227277438454190337548315979217177260934441541003078 67

Summed through 50
The sum is
1.7140062935916160227277438454190337548315979217189574090012146573840 77

Summed through 60
The sum is
1.7140062935916160227277438454190337548315979217189574090012146573952 10

Note that each additional ten elements adds approximately 13 additional digits of precision to the final result. Therefore, there is no reason to perform any additional sums.

3) The third constant of Smarandache

$$\sum_{n=2}^{\infty} \frac{1}{S(2)S(3)\ldots S(n)}$$

is convergent to a number s3, which is between 0.71 and 1.01.

Theorem: The series that defines the third constant of Smarandache is convergent.

Proof: Since S(n) > 2 for all n > 1, we have the inequality

$$\frac{1}{S(2)S(3)\ldots S(n)} < \frac{1}{2^{n-1}}$$

and since

$$\sum_{n=2}^{\infty} \frac{1}{2^{n-1}}$$

is convergent, we can apply the comparison test to conclude that

$$\sum_{n=2}^{\infty} \frac{1}{S(2)S(3)\ldots S(n)}$$

is also convergent.

The program was then modified to sum this series, and the values for the first seven summations, where each increases by ten are as follows.

Summed through 10
The sum is
0.719960317460317460317460317460317460317460317460317460317460317460317

Summed through 20
The sum is
0.719960700043707577861590343776200575060469895596928866376996506241186

Summed through 30

The sum is
0.719960700043708022671712890452749975301490455301794690655548136995178

Summed through 40
The sum is
0.719960700043708022671712902840784494028018774311684748338209707400919

Summed through 50
The sum is
0.719960700043708022671712902840784494134124905674567117249298669322475

Summed through 60
The sum is
0.719960700043708022671712902840784494134124905674892601915153334287983

Summed through 70
The sum is
0.719960700043708022671712902840784494134124905674892601915153400420881

Each increase of ten adds approximately eight digits of accuracy to the sum, so we know that the sum will be less than 0.72, a significant improvement from the range listed in the statement of the problem.

4) The fourth constant of Smarandache

$$\sum_{n=2}^{\infty} \frac{n^{\alpha}}{S(2)S(3)\ldots S(n)}$$

where $\alpha \geq 1$, is convergent to a number s4.

In this case, the statement is not quite accurate, because the series would define a set of constants, one for each value of α. However, it is easy to prove that the series converges for all values of α.

Theorem: The series

$$\sum_{n=2}^{\infty} \frac{n^{\alpha}}{S(2)S(3)\ldots S(n)}$$

converges for all values of $\alpha \geq 0$.

Proof: Since $2 \leq S(n) \leq n$ for all n, we have

$$\frac{n^{\alpha}}{S(2)S(3)\ldots S(n)} \leq \frac{n^{\alpha}}{2^{n-1}}$$

and since

$$\sum_{n=2}^{\infty} \frac{n^{\alpha}}{2^{n-1}}$$

converges,

$$\sum_{n=2}^{\infty} \frac{n^{\alpha}}{S(2)S(3)\ldots S(n)}$$

does as well.

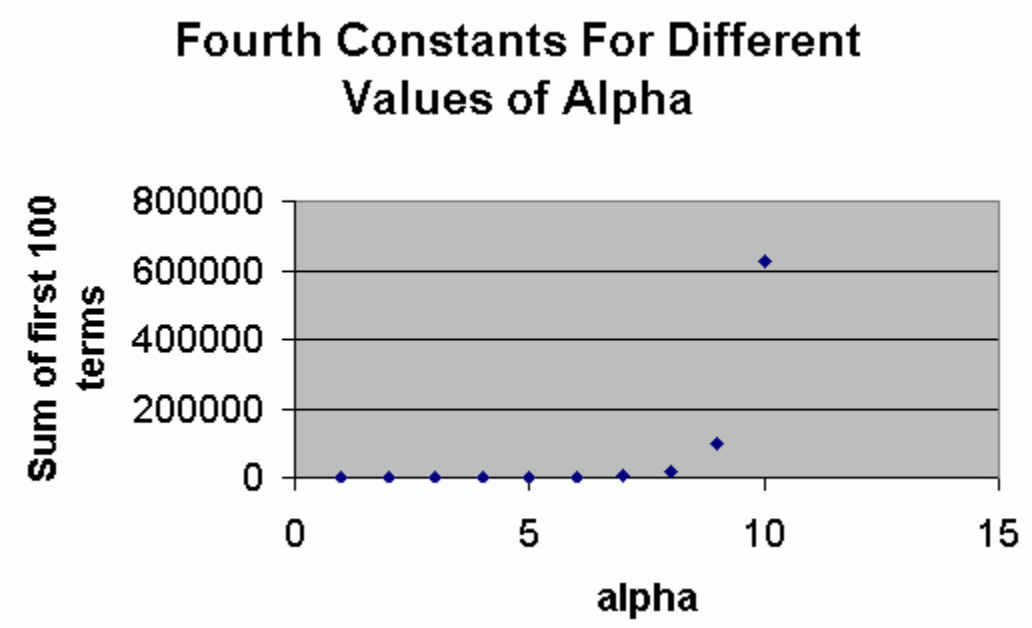
The question then becomes, what is the relative values of the sums as α gets larger? For the purposes of programming simplicity, we will restrict our examination to integral values for α. The partial sums for the first 100 terms of the sequences for α = 1 through 10 were computed and the results are summarized in table 2 and plotted in figure 2. The

multiplying factor in the third column is the ratio of the sums for alpha=(n+1) and alpha=n.

Table 2

Alpha	Sum	Multiplying factor
1	1.728758	----------
2	4.502512	2.604478
3	13.01114	2.889752
4	42.48184	3.265035
5	158.1055	3.721718
6	669.5549	4.234863
7	3195.852	4.773099
8	16970.69	5.310224
9	98953.55	5.830851
10	626338.7	6.329623

Figure 2



From this data, it is clear that the increase in the sum is at least exponential in form. While no attempt will be made here to determine the exact form of the function, it would be interesting to determine what the precise behavior is.

5) The series

$$\sum_{n=2}^{\infty} (-1)^{n-1} \frac{S(n)}{n!}$$

converges to an irrational number.

It is easy to prove that the series converges.

Theorem: The series

$$\sum_{n=2}^{\infty} (-1)^{n-1} \frac{S(n)}{n!}$$

converges.

Proof: Start with the sequence of elements

$$\frac{S(n)}{n!}$$

and, since $S(n) \leq n$, the terms of this sequence are less than or equal to

$$\frac{n}{(n-1)!}$$

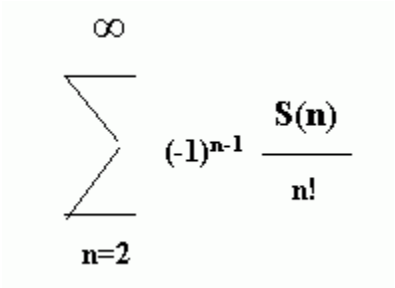
and

$$\lim_{n \to \infty} \frac{S(n)}{n!} = 0.$$

Furthermore,

$$\frac{S(n)}{n!} \geq \frac{S(n+1)}{(n+1)!}.$$

The alternating series theorem can then be used to conclude that the series



is convergent.

To determine what the series sums to, the program was altered to compute a series of partial sums, and the following is a summary of the results.

Summed through 10
The sum is
-0.627861827601410934744268077601410934744268077601410934744268077601 41

Summed through 20
The sum is
-0.627861558367957185446116020782534561535440639518918003801235846490595

Summed through 30
The sum is
-0.627861558367957185318008058414419771773644026319862839588305509842659

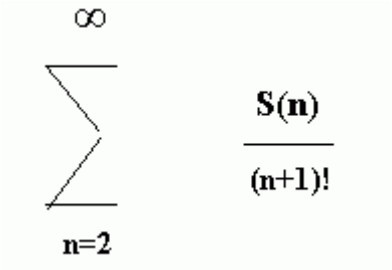
Summed through 40
The sum is
-0.627861558367957185318008058414416030979262054361448063381249484640229

Summed through 50
The sum is
-0.627861558367957185318008058414416030979262054360226720478027692451932

Summed through 60
The sum is
-
0.627861558367957185318008058414416030979262054360226720478027692441121

From these values, and the rapidity with which the digits become stable, it is clear that the sum of the series is very close to that of the last sum.

6) The series



converges to a number s6, where e – 3/2 < s6 < 1.2.

The proof that the series converges is similar to that of the second constant of Smarandache, so it is omitted.

The program was altered in order to compute the values of some partial sums, and the values of the first few are as follows.

Summed through 10
The sum is
0.4993927619448452781786115119448452781786115119448452781786115119448 45

Summed through 20
The sum is
0.4993927857063709807957550372688401042476049130216103481476340604830 64

Summed through 30
The sum is
0.4993927857063709808024456438247344196742738891879714664813773991898 13

Summed through 40
The sum is
0.4993927857063709808024456438247345384466173882001441492373430317767 78

Summed through 50
The sum is
0.4993927857063709808024456438247345384466173882001734627387657353504 74

Summed through 60
The sum is
0.4993927857063709808024456438247345384466173882001734627387657353506 88

From the rapid rate in which significant digits are added, it is clear that the last partial sum is very close to the sum of the series. This value is within the bounds stated in the theorem.

7) The series

$$\sum_{n=r}^{\infty} \frac{S(n)}{(n+r)!}$$

where r is a natural number converges to a number s7.

The proof of convergence for any r > 0 is easy, and is based on the fact that

$$\frac{S(n)}{(n+r)!} \leq \frac{S(n)}{(n+1)!}$$

for $r \geq 1$.

The rate of convergence will rise rapidly as r gets larger and the sum will drop very fast.

To test these two properties, a program was created that will compute the partial sums for various values of r. Since the series of series 6 determined the partial sum for r = 1, in this step the sums were done for r = 2 through 10 and the results are summarized in table 3 and figure 3.

TABLE 3

r	Sum of first 100 terms	Sum(r-1)/Sum(r)	Ratio(r)-ratio(r-1)
1	0.499393	---------	-----
2	0.114976	4.343456	------
3	0.021761	5.283503	0.940048
4	0.003487	6.240672	0.957169
5	0.000484	7.20872	0.968047
6	5.91E-05	8.184053	0.975334
7	6.45E-06	9.164481	0.980427
8	6.35E-07	10.1486	0.984115
9	5.71E-08	11.13546	0.986865
10	4.71E-09	12.12444	0.988975

The behavior demonstrated in table 3 and figure 3 is exactly what we would expect. As r increases, the sum rapidly goes to zero, with the rate of decline increasing as r gets larger.

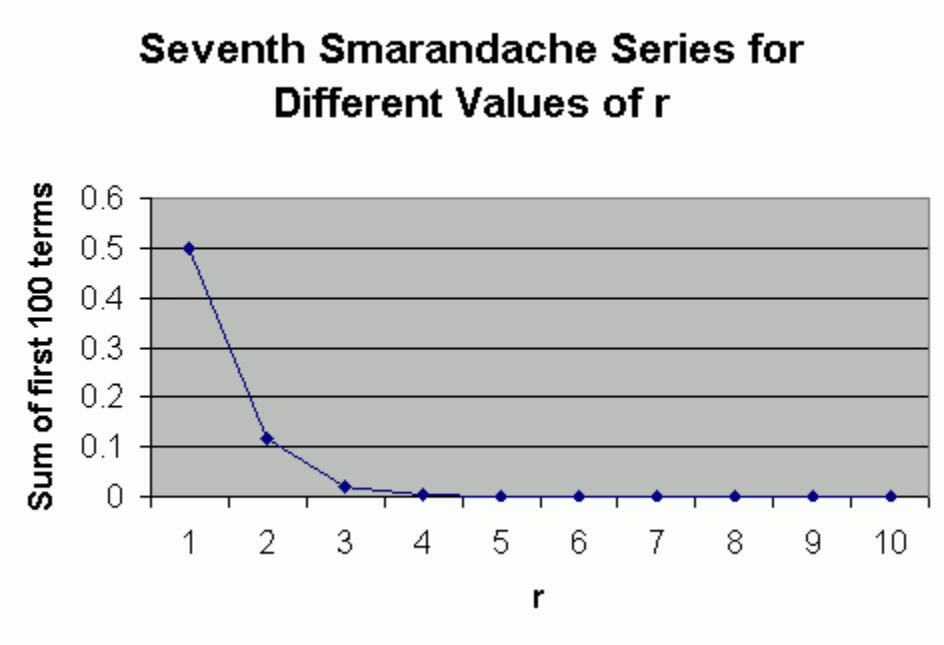
However, while the rate of increase in the rate of decline is fairly constant, it is showing a steady increase as r increases.

There are two interesting questions that can be asked concerning the general behavior of this family of sequences.

Question: Is there a function that will describe the change in the sums of the series as r gets larger?

Question: It appears that the ratio of the sums converges to a number approximately equal to 0.988975 as r increases. Does the ratio of the sums converge and if so, what value does it converge to?

FIGURE 3



8) The series

$$\sum_{n=r}^{\infty} \frac{S(n)}{(n-r)!}$$

where r is a nonzero natural number, converges to a number s8.

The proof of convergence is based on the behavior of the elements as n gets "large enough."

Theorem: The series

$$\sum_{n=r}^{\infty} \frac{S(n)}{(n-r)!}$$

is convergent.

Proof: The first point is that $S(n) \leq n$ for all n, so that

$$\frac{S(n)}{(n-r)!} \leq \frac{n}{(n-r)!}$$

since we can choose a value of $n = k$ such that $k^3 < (k-r)!$ for any fixed value of r and for all $n \geq k$, we have

$n^3 < (n-k)!$

or

$$\frac{n}{(n-k)!} \leq \frac{1}{n^2}$$

Since the largest is known to converge, we can put the inequalities together and conclude that

$$\sum_{n=r}^{\infty} \frac{S(n)}{(n-r)!}$$

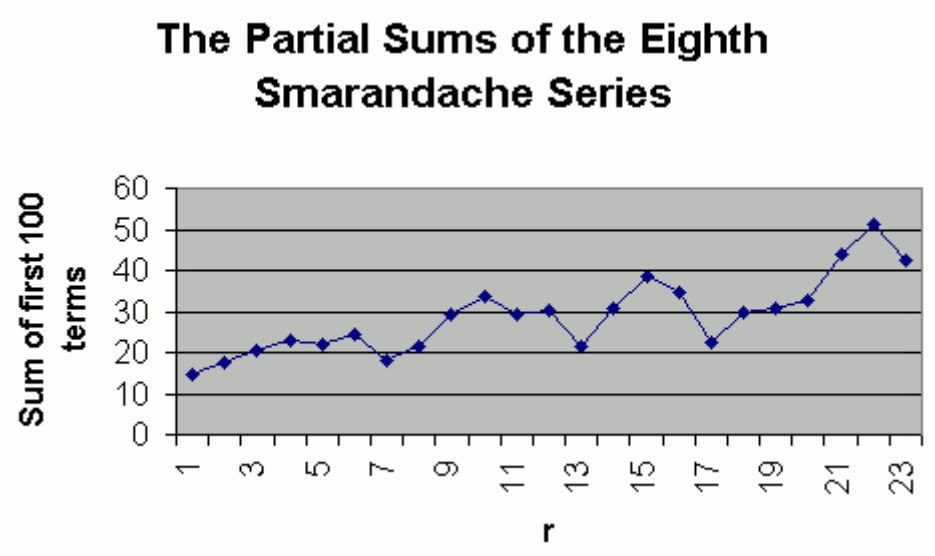
also converges.

To determine the behavior of the members of this family of series a program was run that computes the partial sums for the first 100 terms in the series for r=1, 2, . . . , 14. The results of those program runs are summarized in table 3 and figure 3.

TABLE 3

r	Sum of first 100 terms
1	4.410682
2	8.023568
3	10.33461
4	11.8926
5	12.47475
6	13.30832
7	15.34459
8	14.61925
9	17.77525
10	20.51191
11	22.95029
12	21.73725
13	24.28592
14	18.25034
15	21.34582
16	29.45331
17	33.74969
18	29.32456
19	30.34172
20	21.60343
21	30.7063
22	38.29902
23	34.64206
24	22.54666
25	29.96163
26	30.81374
27	32.71052
28	44.11692
29	51.44522
30	42.61618

FIGURE 3



From the structure of the graph, it appears that the sum is generally increasing in a linear manner, although the values vary considerably from this norm. A little thought is all that is needed to understand this behavior. Since S(p)=p for p a prime, and S(n) is generally substantially smaller than n for composite n, the value of the sum is dramatically altered if the values or r and the numbers slightly larger than r are prime.

For example if r is 29, then the first three terms of the series are

$$\frac{29}{0!}, \frac{5}{1!}, \frac{31}{2!}$$ or 29, 5 and 15.5

however, if r is 24, the first three terms are

$$\frac{4}{0!}, \frac{10}{1!}, \frac{13}{2!}$$ or 4, 10 and 6.5.

Since the bulk of the value of the sum is contributed by the first few terms when the denominator is small, this explains the fluctuations of the graph.

The general increasing trend is due to the fact that beyond the initial terms, those that are in common would tend to be divided by a smaller factorial. For example, the term when n=30 when r=5 would be

$$\frac{5}{25!}$$

and when r=10, the term when n = 30 is

$$\frac{5}{20!}$$

Clearly, the general increase in the values as r gets larger would be very small, hence the appearance of a linear increase.

Theorem: There is no upper limit to the sums of the elements of the family of series

$$\sum_{n=r}^{\infty} \frac{S(n)}{(n-r)!}$$

Proof: Assume that there is an upper limit and call it M. By the infinitude of the primes, we can find a prime p such that p > M. Let p = r and consider the first term of the series

$$\frac{S(p)}{(p-p)!} = \frac{p}{1} = p > M,$$

contradicting the choice of M. Therefore, there is no upper limit.

Question: Is the general trend of increasing sums of the elements of the family of series linearly increasing as r increases?

9) The series

$$\sum_{n=2}^{\infty} \frac{1}{\sum_{i=2}^{n} \frac{S(i)!}{i}}$$

is convergent to a number s9.

It is easy to verify that this series converges. Since S(p) = p for p a prime, the prime elements of the sum in the denominator are p!/p, which get large very quickly. Therefore, the sums in the denominator grow rapidly and the series is convergent.

The next question is to determine what the sum of the series is. To obtain a partial answer to this question, a computer program was written to compute a series of partial sums of the series and the results are listed below.

Constant sum 9 of Smarandache
Summed through 20
The sum is
1.509179662100384467866157437047505079488430968376730331596977696154959

Constant sum 9 of Smarandache
Summed through 30
The sum is
1.509179662100384779238073500490370856013156675656663786691622809731299

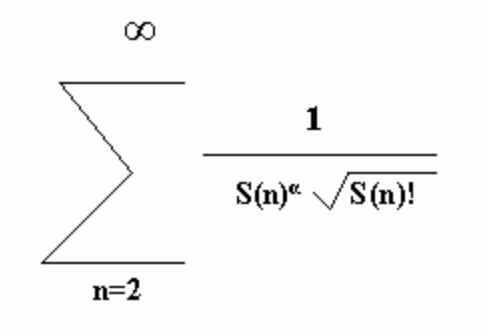
Constant sum 9 of Smarandache
Summed through 40
The sum is
1.509179662100384779238073500490393449968876195164764988973089531110085

Constant sum 9 of Smarandache
Summed through 50
The sum is
1.509179662100384779238073500490393449968876195167219068044865898945433

Constant sum 9 of Smarandache
Summed through 60
The sum is
1.509179662100384779238073500490393449968876195167219068045229361932962

An accurate estimate for the sum of the series can be extracted from this series of partial sums.

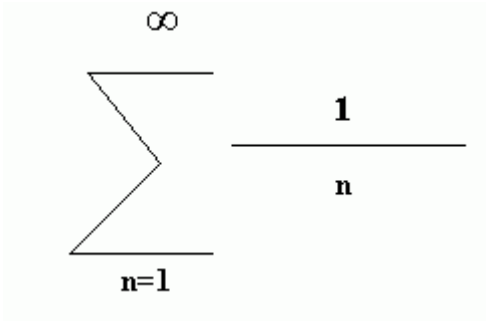
10) The series



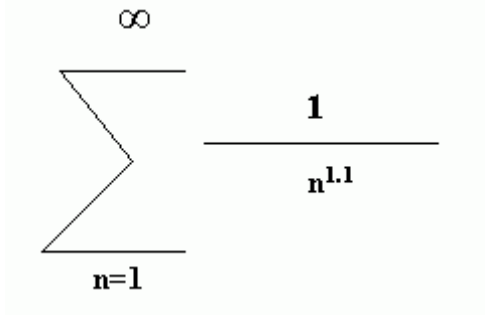
where $\alpha > 1$, is convergent to a number s10.

When I first examined this series, the limits on α were immediately suspect. While S(n) does grow slowly, the factorial function grows very rapidly, so rapidly that it overwhelms the square root function. Furthermore, the factorial, even after taking the square root, will dominate $S(n)^{\alpha}$. Therefore, as n gets larger, that square root term of the denominator will be much larger that S(n) raised to any power.

To test this hypothesis, a program was written that will compute the first partial sums for this series, the harmonic series



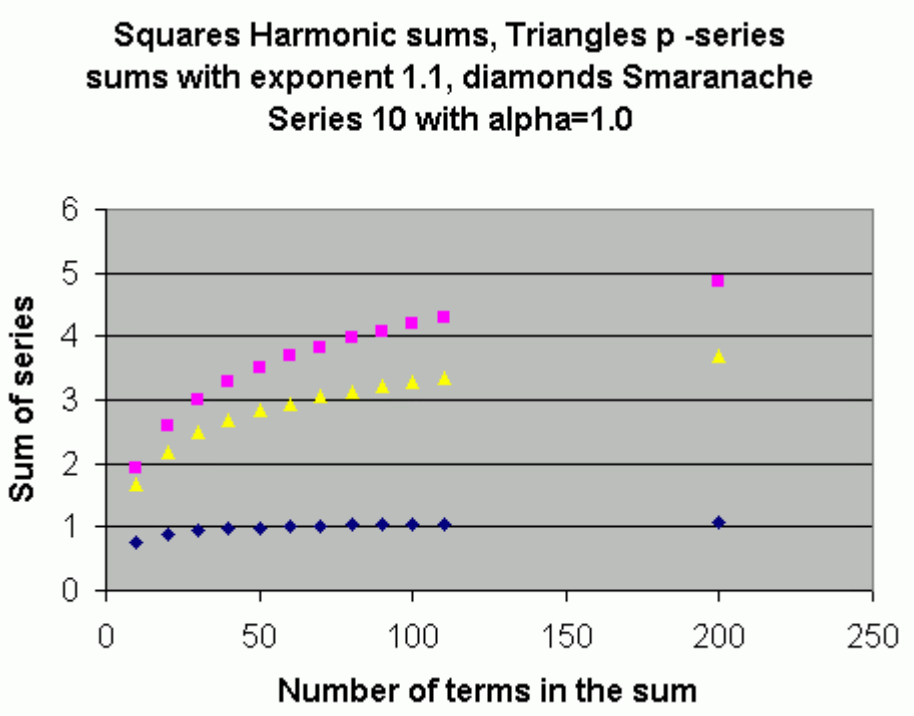
and the p-series with p = 1.1.



The harmonic series is known to diverge and the p series with p=1.1 is known to converge.

A series of partial sums for each series was computed and the results are summarized in figure 4.

Figure 4



The plot with squares are the partial sums of the harmonic series, the plot of triangles the partial sums of the p-series and the diamonds the partial sums of the Smarandache series with alpha = 1. From this plot, there are strong indications that the Smarandache series is convergent for alpha = 1. Therefore, the following conjecture is offered.

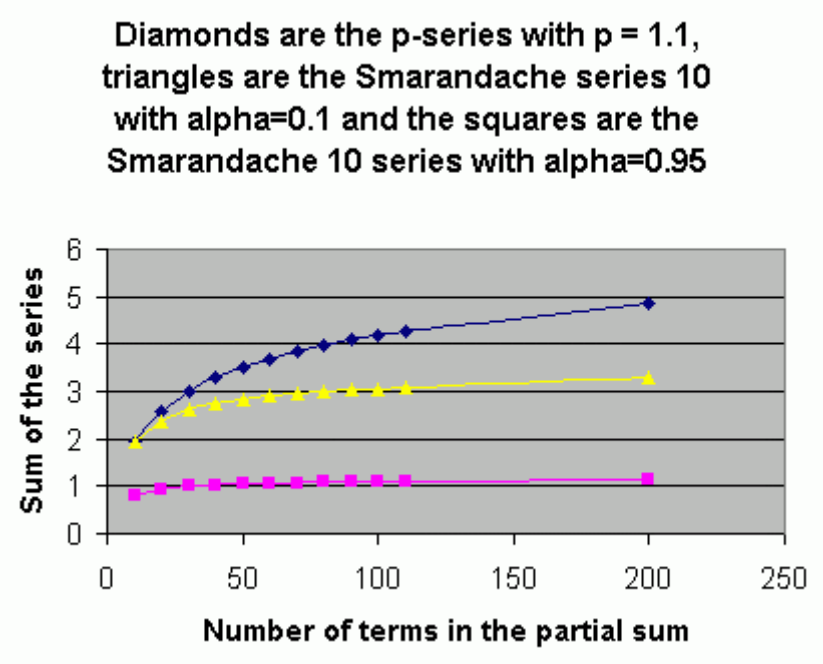
Conjecture: The series

$$\sum_{n=2}^{\infty} \frac{1}{S(n)^{\alpha}\sqrt{S(n)!}}$$

converges for $\alpha = 1$.

With the previous results and conjecture in mind, the next question to consider is if the series is convergent for any values of $\alpha < 1$. To test this, the program was rerun for $\alpha = 0.95$ and $\alpha = 0.01$ and the results appear in figure 5.

Figure 5



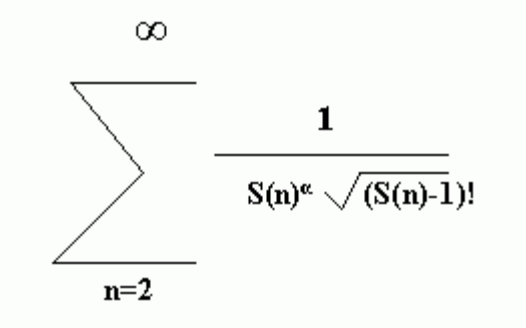
Once again, the behavior of the partial sums strongly indicates that the Smarandache series with $\alpha = 0.95$ and $\alpha = 0.1$ converge. This leads to the following question.

Question: Does the series

$$\sum_{n=2}^{\infty} \frac{1}{S(n)^{\alpha}\sqrt{S(n)!}}$$

converge for all values of $\alpha > 0.0$?

11) The series



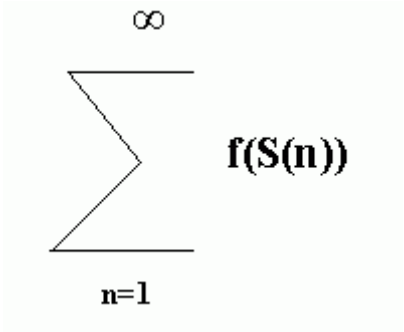
where $\alpha > 1$, is convergent to a number s11.

The general behavior of this series would be similar to that for series number 10, although more slowly, so no additional analysis will be done.

12) Let f: $N \rightarrow R$ be a function defined by

$$f(n) = \frac{c}{n^{\alpha}\,(d(n!)) - d((n-1)!)}$$

for $n > 0$, d(n) the number of divisors function and $\alpha > 1$ and $c > 1$ constants. Then the series



is convergent to a number s12, where the sum will be different for each choice of α and c.

To begin, the function f(n) is not defined for $n = \{0,1\}$, as 1! and 0! are both 1, leading to a zero in the denominator. Therefore, when a program was created to test this series, the summation was begun for $n = 2$.

Provided that it is not zero, the value of the constant c in the numerator has no bearing on whether the series converges or diverges. Therefore, we will simply take $c = 1$ and adjust any convergent value accordingly.

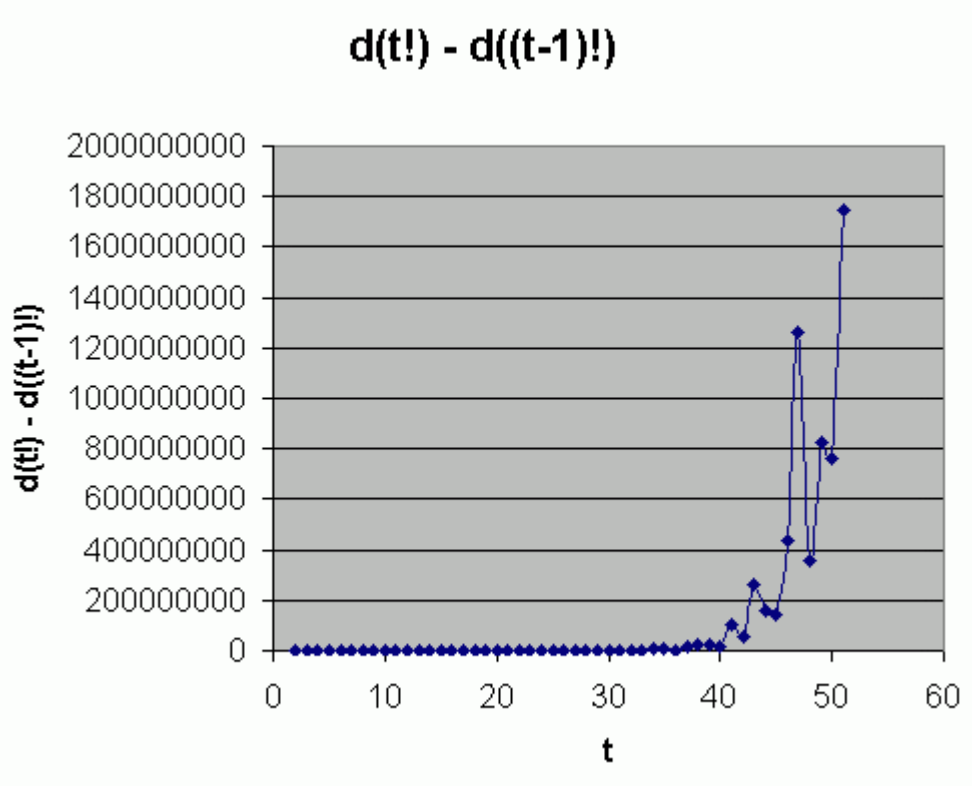
When looking at the function, the difference $(d(n!)) - d((n-1)!)$ appears to be the key segment. If p is prime, then p! and (p-1)! will share all factors other than p, and by the formula used to compute d(x), if $d((p-1)!) = k$, then $d(p!) = 2k$. Therefore, the difference would be $2k - k = k$, which for factorials, grows very quickly.

To examine this in more detail, a program was written to plot the growth of the function

$$(d(n!)) - d((n-1)!)$$

for $n > 1$. The results of this computation are given in figure 6.

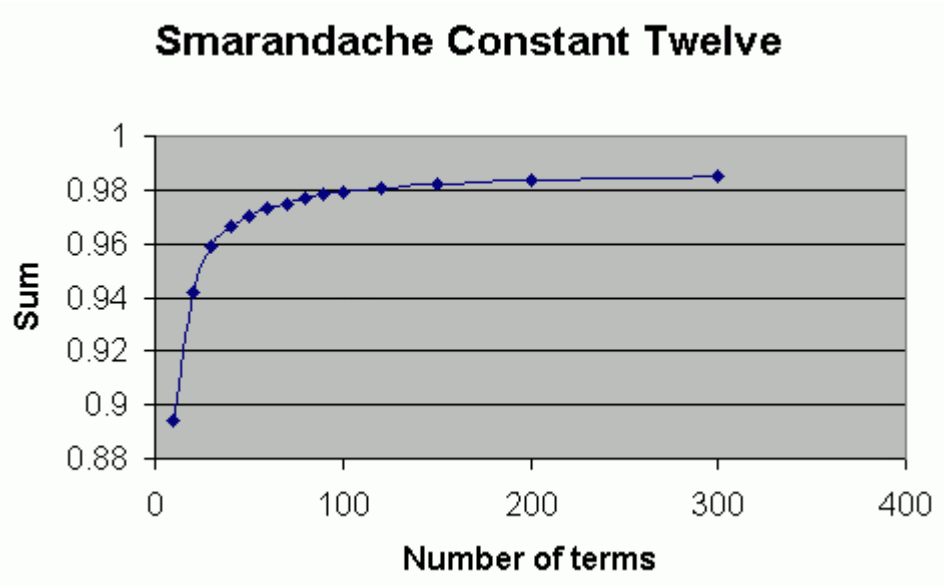
Figure 6



Note that while there is some variation, the overall trend of the graph is to move upward in an exponential manner.

Given this result, it would appear that convergence of the series using f(n) would be dictated by the difference in the denominator rather than the t^{α}. Therefore, a program was written to determine the values of some partial sums of the series for $\alpha = 1$ and the results appear in figure 7.

Figure 7



The behavior of the partial sums clearly indicates that the series converges for $\alpha = 1$. This leads to an additional and more general question.

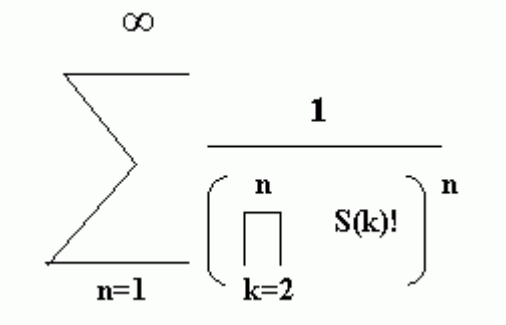
Question: For what values of $\alpha > 0$, is the series

$$\sum_{n=2}^{\infty} f(S(n))$$

convergent?

Given the apparent rapid growth of the difference $(d(n!)) - d((n-1)!)$, we conjecture that the series is convergent for all $\alpha > 0$.

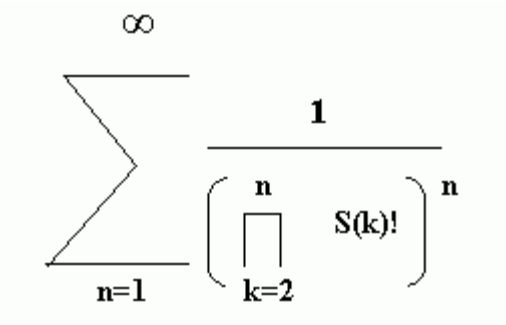
13) The series



is convergent to a number s13.

Note: As stated, the series definition is no doubt incorrect. The product in the denominator starts at k = 2, while the sum starts at n = 1. Since S(1) = 1 and $1^n = 1$, we will change the starting value in the product to k =1 with no change in the product.

Theorem: The series

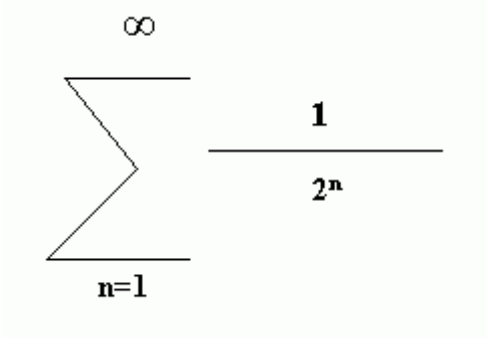


is convergent.

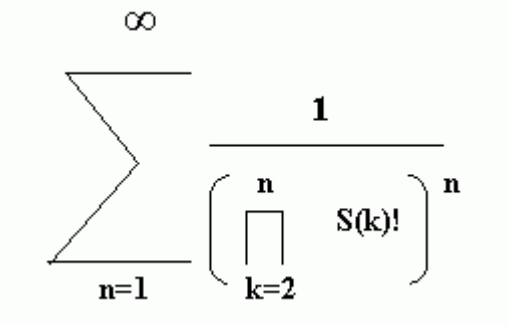
Proof: It is easy to see that

$$\frac{1}{\prod_{k=2}^{n} S(k)!} \leq \frac{1}{2^n},$$

therefore, since



is convergent, it follows that



must also be convergent.

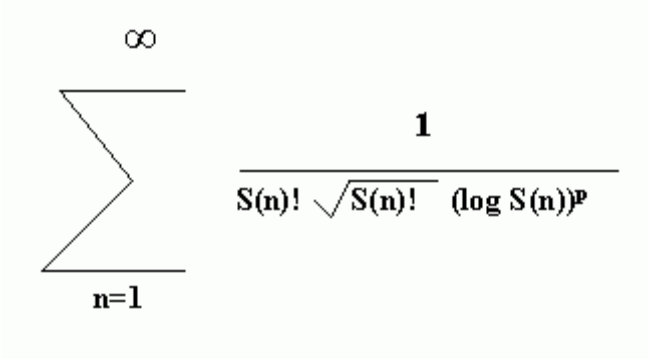
To obtain an estimate for the value of the sum, a computer program was written to compute a series of partial sums, and the results are as follows.

Constant sum 13 of Smarandache
Summed through 10
The sum is
1.250578703849058728304585135878624611170067441801003476191155937686560

Constant sum 13 of Smarandache
Summed through 20
The sum is
1.250578703849058728304585135878624611170067441801003476191155937686560

The rapid convergence is certainly expected, the denominator is composed by taking the exponent of a product of factorials.

14) The series



where p > 1, is convergent to a number s14.

The first point to note is that S(1)=1 and log 1 = 0, so the definition above needs to be modified to start at n = 2 rather than n = 1.

In examining the denominator expression, it is clear that the first two components can be combined to form

$$(S(n)!)^{3/2}$$

and that this component will dominate the remaining

$$(\log S(n))^p$$

for large values of n independent of the value of p. Therefore, the convergence is independent of the value of p. Furthermore, all logarithmic functions have a similar behavior, so in our testing, we are free to choose the type of logarithm. In this case, we will use the logarithm to the base 10.

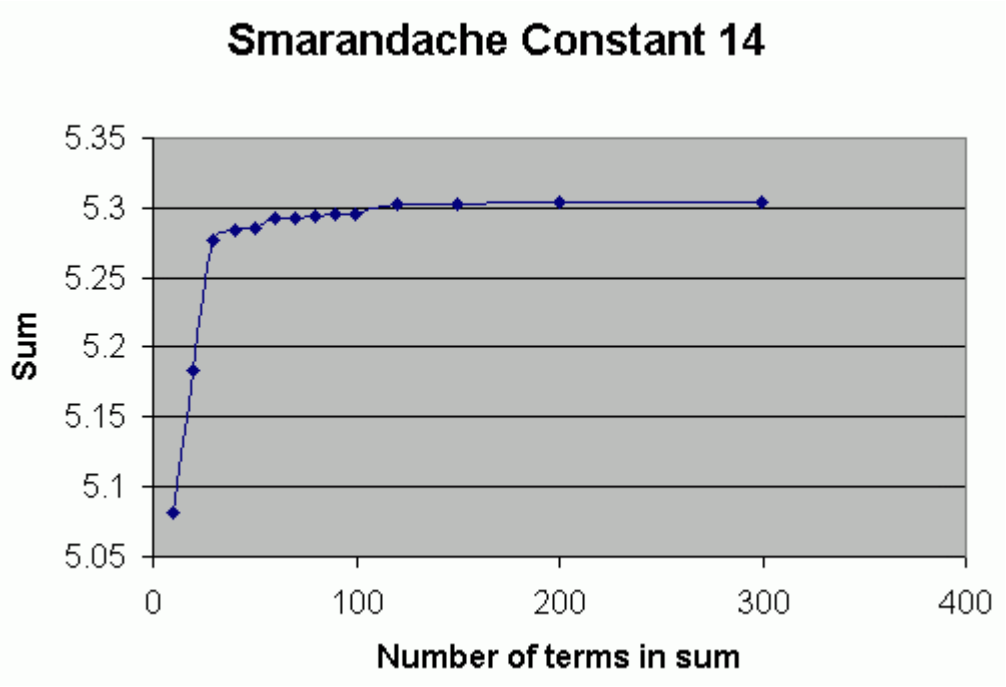
Since S(n) > 2 for n > 2, for our testing purposes, we can use the constant 0.10 for the value of

$$(\log S(n))^p$$

the actual values will be larger.

A program was then created to compute a series of partial sums of this series using 0.10 as the value of the third component of the denominator. The results of several runs are summarized in figure 8.

Figure 8



From the behavior of these partial sums, it is strongly indicated that the series converges. Note once again that the actual behavior of the series will be to grow more slowly than this graph indicates. We have taken the smallest value for

$$(\log S(n))^p$$

to be 0.1 and used it for all terms rather than the actual value, which will slowly increase. This increase in the value of the denominator would make the terms smaller than what they are in these partial sums.

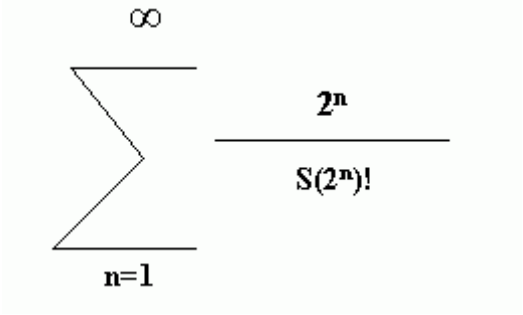
15) The series

$$\sum_{n=1}^{\infty} \frac{2^n}{S(2^n)!}$$

is convergent to a number s15.

It is easy to prove that the series is convergent.

Theorem: The series



is convergent.

Proof: By definition of the Smarandache function, $S(2^n) = m$ is the smallest number such that the list of numbers

2, 3, 4, 5, . . . , m

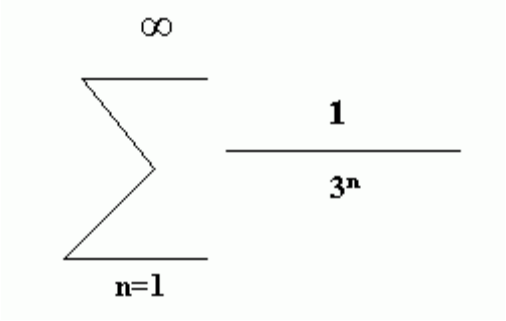
has n instances of 2 as a factor. If we then take the factorial of m, that product would have n instances of 2 as well as all odd prime factors of the numbers from 3 through m. If the product of all the odd prime factors of the numbers through m is denoted by OddPrimeProduct(m), it follows that

$$\frac{2^n}{S(2^n)!} = \frac{2^n}{2^n \text{ OddPrimeProduct}(m)} = \frac{1}{\text{OddPrimeProduct}(m)}.$$

Since

$$\frac{1}{3^n} > \frac{1}{\text{OddPrimeProduct}(m)}$$

and



is convergent, it follows that the series is.

A program was written to determine the values of some partial sums of this series, and the results are as follows.

Constant sum 15 of Smarandache
Summed through 10
The sum is
1.527851531184864518197851531184864518197851531184864518197851531184864

Constant sum 15 of Smarandache
Summed through 20
The sum is
1.527851557623860671249652664757454036050476294523288703169328585028781

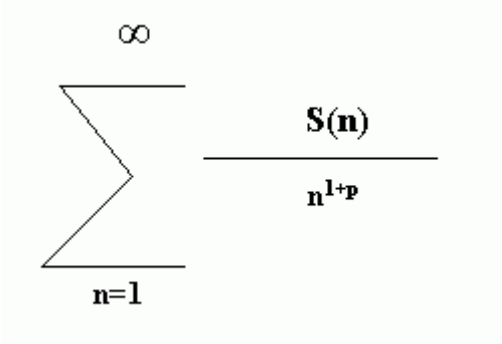
Constant sum 15 of Smarandache
Summed through 30
The sum is
1.527851557623860681410794380787355482609395803978747443916738964151399

Constant sum 15 of Smarandache
Summed through 40
The sum is
1.527851557623860681410794388963249556705080221130858847199420796016306

Constant sum 15 of Smarandache
Summed through 50
The sum is
1.527851557623860681410794388963249556705081049178753714969083597611545

From these values, it is easy to determine an approximate value for the sum of the series.

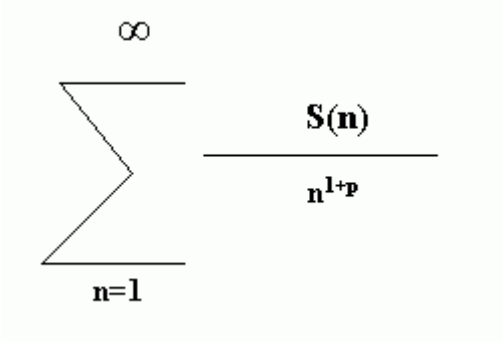
16) The series



where $p > 1$ is a real number, converges to a number s16. (For $0 \leq p \leq 2$, the series diverges.)

The problem is incorrect as stated, for the series is easily seen to be convergent if $p > 1$ and divergent if $p \leq 1$.

Theorem: The series



a) Is convergent if $p>1$.

b) Is divergent if $0 \leq p \leq 1$.

Proof:
a) Since $S(n) \leq n$,

$$\frac{S(n)}{n^{1+p}} \leq \frac{1}{n^{p}}$$

and the series formed from the terms on the right is convergent when $p > 1$, it follows that the series formed from the terms on the left is also convergent.

b) Since $S(q) = q$ for q prime, we have

$$\frac{S(q)}{q^{1+p}} = \frac{q}{q^{1+p}} = \frac{1}{q^{p}}$$

Since the sum of the reciprocals of the primes diverges when $p \leq 1$, the series

$$\frac{S(q)}{q^{1+p}}$$

diverges, so all series in which it is embedded also diverge.

References

1 .http://www.gallup.unm.edu/~smarandache/SNAQINT.txt
2. http://www.gallup.unm.edu/~smarandache/SNAQINT2.TXT
3. http://www.gallup.unm.edu/~smarandache/SNAQINT3.TXT

Index

Florentin Smarandache is truly a Renaissance man, having produced quality work in art, mathematics and literature. This is my fifth book where I expand on some of his mathematical ideas and there seems to be no end in sight.

In chapter 1, several sequences created by concatenating all of the natural numbers are examined. By expressing the numbers in bases other than 10, a single type of concatenation can generate several different sequences. Upon examination, it can be seen that there are some significant differences between the sequences.

Subsets of the natural numbers can be concatenated to create additional sequences and several are examined in chapter 2. The successive odd numbers and the successive even numbers are concatenated to form two of the sequences that are considered in detail.

A stereogram is a two-dimensional image that contains a three-dimensional image. By looking at it long enough, the eyes lose focus in just the right way and the image appears. Simple Smarandache stereograms can be constructed using only characters and many different images are presented in chapter 3.

The Smarandache functions can be used in combination with other mathematical operations to create many different infinite series. Many such series are examined in chapter four and conclusions whether they converge or diverge are reached.

ISBN 1-931233-23-3
52995>
9 781931 233231